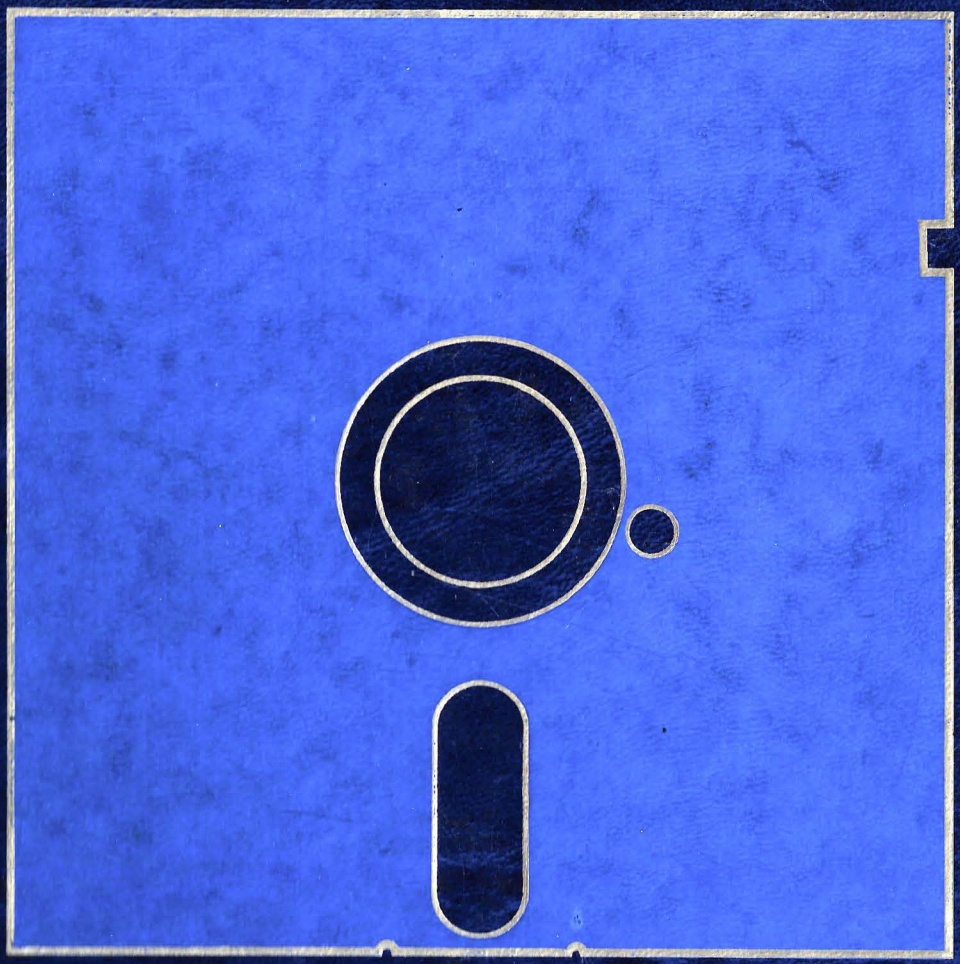
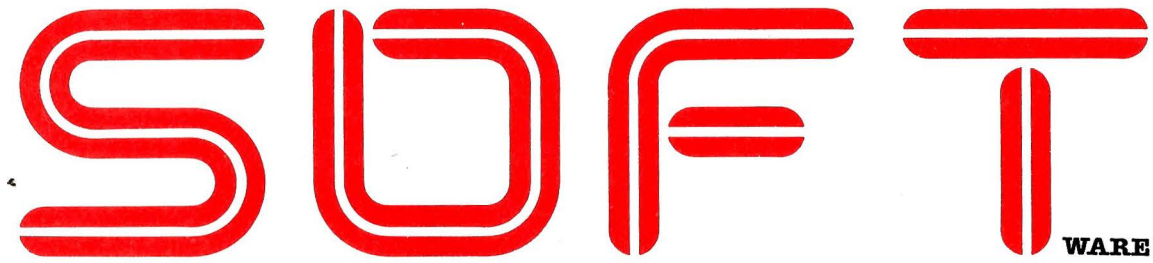
SOFTWARE





Nueva Lente/Ingelek

una publicación

EDICIONES NUEVA LENTE, S. A.,
y EDICIONES INGELEK, S. A.

Director-Editor:
Por NUEVA LENTE, S. A.:
MIGUEL J. GOÑI
Por INGELEK, S. A.:
ANTONIO M. FERRER
Director de producción:
RICARDO ESPAÑOL
Jefe de producción:
SANTOS ROBLES

Director de la obra:
FRANCISCO LARA (PL/3)
Colaboradores:
MANUEL MUÑOZ
DAVID SANTAOLALLA
SANTIAGO RUIZ
LUIS COCA
MIGUEL ANGEL VILA
MIGUEL ANGEL SANCHEZ
VICENTE ROBLES

Diseño e ilustración:
JOSE OCHOA (PL/3)
Maquetación:
JUAN JOSE DIAZ SANCHEZ
Fotografía:
(Equipo Gálata)
EDUARDO AGUDELO y
ALBINO LOPEZ
Redaccion:
PL/3 calc

© Ediciones Nueva Lente, S. A. y
Ediciones Ingelek, S. A.
Madrid, 1984
Dirección, Redacción y
Administración:
Benito Castro, 12. 28028 Madrid
Tels. 245 45 98 y 246 73 67

ISBN, tomo primero: 84-7534-103-9
ISBN, tomo segundo: 84-7534-104-7
ISBN, tomo tercero: 84-7534-105-5
ISBN, tomo cuarto: 84-7534-106-3
ISBN de la obra: 84-7534-101-2

Fotomecánica:
OCHOA, S. A.
Ricardo Ortiz, 74. 28017 Madrid
Impresión:
GRAFICAS REUNIDAS, S. A.
Avda. de Aragón, 56. 28027 Madrid
Depósito legal: M. 41.955-1984
PRINTED IN SPAIN

Queda prohibida la reproducción total o parcial de esta obra sin permiso escrito de los Editores.

Variables suscritas

Conjuntos de variables de múltiples dimensiones

Para trabajar en BASIC con datos que no poseen un valor fijo, hay que recurrir a las variables. Como ya sabemos, su cometido no es otro que "guardar" o memorizar los datos que les sean asignados. Así, por ejemplo, en la variable MEDIA se puede almacenar la nota media por alumno de un determinado curso. Si se quiere diferenciar las notas medias de los distintos cursos habrá que utilizar más variables. Por ejemplo: MEDIA1, MEDIA2, ... A partir de todas ellas, se puede calcular la nota media total de los cursos de EGB. La forma de hacerlo es sencilla y bastaría con una sola línea de programa:

```
50 LET MEDEGB=(MEDIA1+MEDIA2+
MEDIA3+MEDIA4+MEDIA5+MEDIA6+
MEDIA7+MEDIA8)/8
```

No cabe duda que esta forma de trabajar con conjuntos resulta tediosa. El mayor inconveniente radica en la necesidad de acceder a cada dato individualmente. Ello puede resolverse en parte haciendo uso de un nuevo tipo de variables: las variables de conjunto, o variables suscritas.

En el ejemplo anterior se utilizaban ocho variables distintas para almacenar datos de características similares. Sus nombres sólo se diferenciaban en el último carácter, lo cual denotaba una naturaleza común.

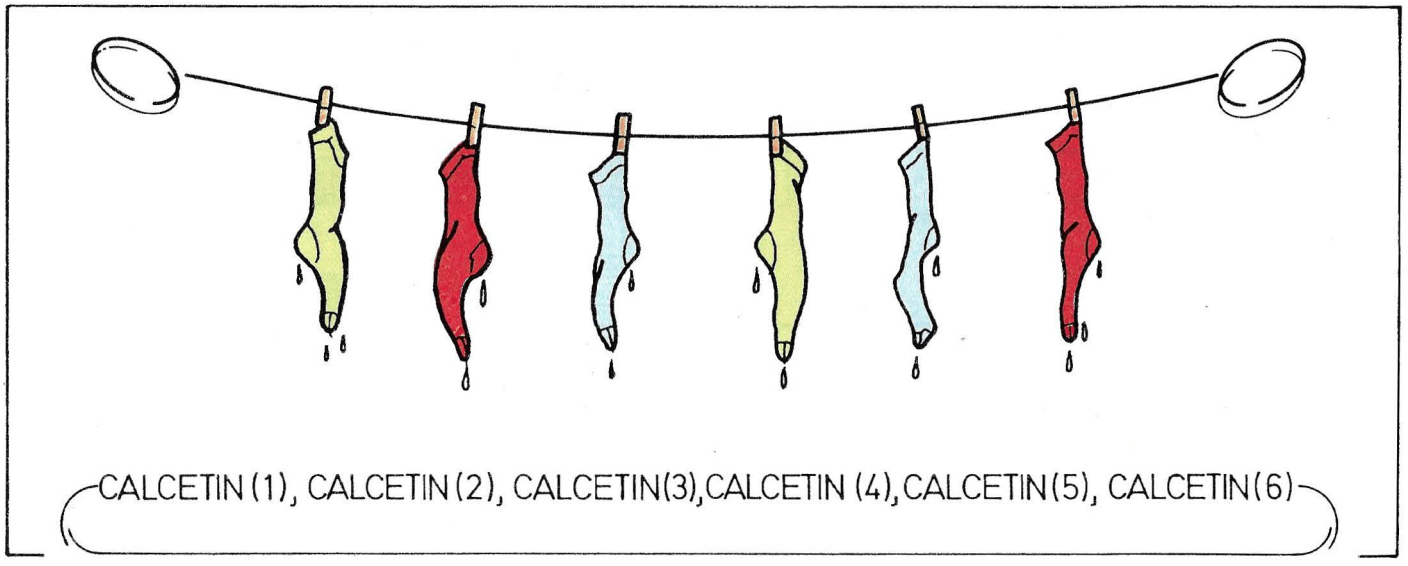
Las ocho variables pueden considerarse como un conjunto homogéneo de datos, al que convendría dar un nombre genérico. Los elementos del conjunto debieran ser accesibles por medio de dicho nombre, acompañado por un indicativo para precisar el elemento elegido. Estas son, precisamente, las características de una variable de conjunto o "array" en BASIC.

En notación matemática, los conjuntos de datos (por ejemplo, coeficientes de un polinomio) se suelen expresar por medio de un nombre genérico y un subíndice indicativo de cada elemento. En BASIC, el método es similar: un nombre de variable y un índice encerrado entre paréntesis. El conjunto de variables del ejemplo inicial podría expresarse de la siguiente forma:

```
MEDIA(1),MEDIA(2) ...MEDIA(8)
```

El nombre genérico del conjunto es MEDIA; éste va acompañado por el índice adecuado en cada caso. Tal como se observa, los índices son numéricos; más concretamente, números naturales. Este hecho proporciona algunas ventajas. En primer lugar, los índices definen un orden dentro del conjunto. Orden que puede reflejar una sucesión lógica de los datos (por ejemplo, la sucesión de las calificaciones académicas a lo largo de los sucesivos meses).

La segunda ventaja radica en la posibilidad de utilizar una segunda variable para almacenar el valor del índice. En efecto, los índices son datos numéricos, y como tales pueden ser tratados de forma mate-



Una variable suscrita o "array" de una sola dimensión, puede representarse como una línea de elementos que obedecen a un mismo nombre de variable y que se identifican por su respectivo índice.

Algunos dialectos del lenguaje BASIC permiten optar por el valor 0 ó 1 como primer índice de los elementos de una variable suscrita. Tal es el caso del BASIC que suele equipar al IBM-PC y a los ordenadores compatibles con el mismo.

mática. El siguiente ejemplo refleja un método válido para acceder al cuarto elemento de un "array" o conjunto de variables suscritas:

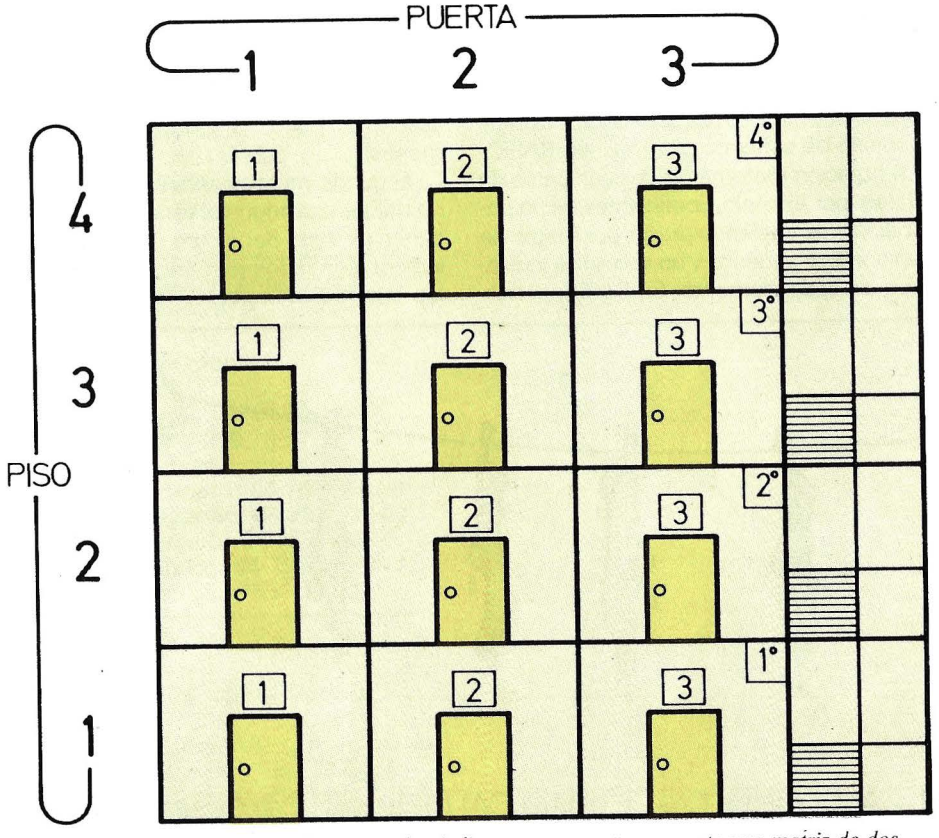
```
30 LET INDICE=4
40 PRINT MEDIA(INDICE)
```

Esta característica otorga una potencia y flexibilidad importantes a las variables de conjunto. Así, por ejemplo, las operaciones con "arrays" se pueden realizar de forma recursiva. Esto es: definiendo la operación para un elemento genérico y haciendo variar el índice para cubrir todos los elementos del conjunto. La presentación en la pantalla puede ordenarse, en consecuencia, con extremada sencillez:

```
120 FOR INDICE=1 TO 8
130 PRINT MEDIA(INDICE)
140 NEXT INDICE
■
```

En el ejemplo se ha definido la operación a realizar una sola vez (línea 130). El bucle FOR/NEXT será el encargado de "barrer" todos los posibles valores del índice (del 1 al 8), con lo que la operación afectará sucesivamente a todos los elementos del "array".
Adoptando este método, el ejemplo para el cálculo de la media total de los cursos de EGB se simplifica en gran medida:

```
50 LET SUMA=0
60 FOR I=1 TO 8
70 LET SUMA=SUMA+MEDIA(I)
80 NEXT I
90 MEDEGB=SUMA/8
■
```



Cuando un conjunto de variables posee dos índices, nos encontramos ante una matriz de dos dimensiones. Un edificio de varios pisos, con varias puertas por planta, constituye un ejemplo ilustrativo de matriz de dos dimensiones. Cada apartamento es identificado por un par de índices: piso, puerta.

Almacenamiento en memoria de variables suscritas

Un conjunto de elementos, como ya se ha comentado, puede almacenarse en memoria bajo un mismo nombre de variable suscrita. Para distinguir a un elemento de otro es necesario recurrir a una serie de números, separados por comas, que reciben el nombre de "índices". Estos acompañan al nombre de la variable e identifican a cada uno de los elementos.
Cuando se define una estructura de este tipo, se indica el número de elementos distintos que van a residir en memoria bajo el mismo nombre genérico. En general, el acceso a cada posición de memoria se realiza a través de un número que identifica su emplazamiento o "dirección" dentro del espacio de memoria. Cuando el ordenador desea conocer el valor de una variable, le basta con examinar el contenido de la posición de memoria asociada a dicha variable; dependiendo del tipo de variable de que se trate, leerá, a partir de esa posición, el contenido de otras posiciones contiguas.
En el caso de los conjuntos de variables o "arrays", la cosa no es tan sencilla. La existencia de más de un elemento bajo el mismo nombre genérico crea alguna dificultad. Si bien, es evidente que a partir del nombre de la variable y del valor de los índices, será posible acceder a cualquiera de los elementos del conjunto.
La forma de acceder a los elementos de las variables suscritas no presenta excesivas complicaciones. En primer lugar, consideremos un conjunto de tan sólo una dirección. La variable del conjunto o "array" tendrá asociada una dirección de memoria; dirección a partir de la que estará localizada su zona de almacenamiento. En segundo lugar, hay que precisar si nuestro ordenador comienza a numerar los elementos del "array" a partir del índice 0 ó del 1. Es posible que quepa la elección mediante el comando OPTION BASE. En cualquier caso, en el ejemplo se considerará que la numeración empieza con el índice cero.
Supondremos, asimismo, que i es el índice que corresponde a un determinado elemento del "array", que L es el tamaño en bytes de cada uno de los elementos y que D es la dirección de memoria asociada a la variable de conjunto. La representación gráfica de su almacenamiento es la que refleja el gráfico adjunto (figura a).
Las direcciones de memoria que interesa son:

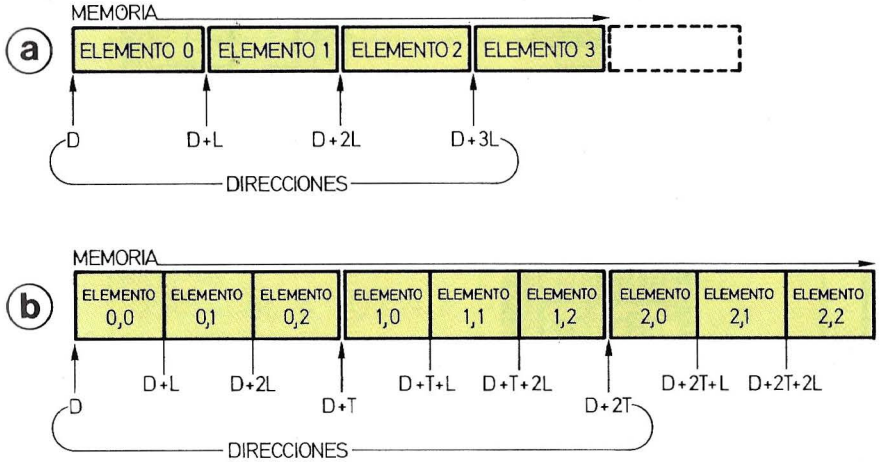
D: para el primer elemento,de índice 0
D+L: para el segundo elemento,de índice 1
D+2L: para el tercer elemento,de índice 2
D+3L: para el cuarto elemento,de índice 3

Sin mayores dificultades, es posible obtener una ecuación que permita calcular las direcciones de cada elemento:

DIR=D+(L*i)

En el supuesto de que el índice no empezara a partir de 0, sino que lo hiciera a partir del valor 1, bastaría con un simple retoque para que la ecuación permitiera obtener la dirección efectiva de cada elemento:

DIR=D+(L*(i−1))



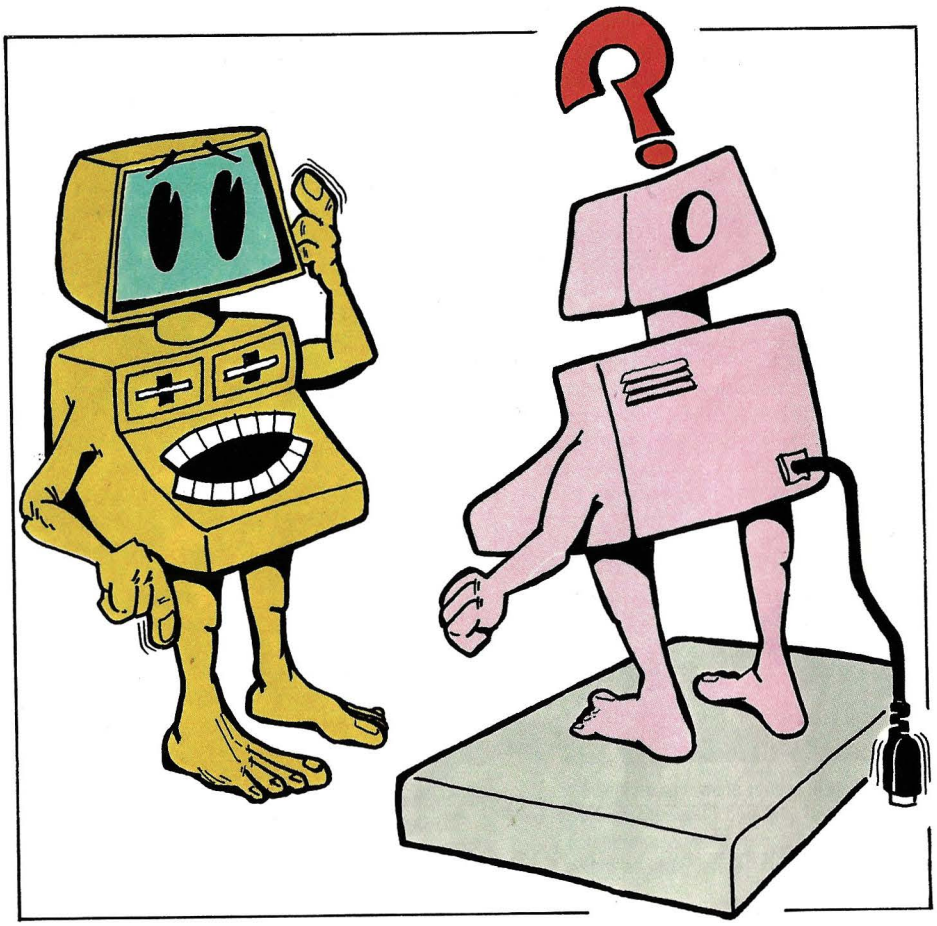
Tal como se observa, el cálculo de la dirección parte de dos componentes. Por un lado, una dirección absoluta (D), coincidente con la dirección inicial de la zona de almacenamiento reservada a la variable suscrita; y por otro, de un valor relativo que indica la distancia al elemento desde el comienzo de la zona de almacenamiento reservada: L*(i-1). En efecto, este valor depende del índice propio de cada elemento del "array".
¿Qué sucede cuando se tiene más de una dimensión? La complicación no es excesiva. Se trata de añadir un nuevo valor de desplazamiento relativo a la ecuación. Imagine que se trata de un libro (la variable suscrita), cuyo primer índice indica el número de página, mientras que el segundo identifica a cada palabra de la misma. Para localizar la posición de una determinada palabra, es necesario conocer a cuantas palabras de distancia del comienzo de la página se encuentra.
Veamos, por ejemplo, cómo se almacena en la memoria un "array" coincidente con una matriz de dos dimensiones (índices i y j) con 3×3 elementos (ver figura b). En esta ocasión, T es el tamaño del subconjunto de todos los elementos que tienen el primer índice común. En consecuencia, si el número de elementos para ese índice es N, el valor de T será igual a N*L.
El cálculo de la dirección de comienzo de cualquier elemento se reduce, pues, a la siguiente expresión:

DIR=D+(i*T)+(j*L)

Aplicado al ejemplo de la figura b, se observa que, en efecto, la posición de memoria ocupada por el elemento 1,2 coincide con:

DIR=D+(1*T)+(2*L)=D+T+2L

En el cálculo de la dirección efectiva de cada elemento intervienen, pues, tres sumandos. El primero (D) coincide con la dirección absoluta a partir de la que comienza la zona de almacenamiento del "array". El segundo es un desplazamiento relativo al primer índice, que nos coloca al principio de la porción de memoria ocupada por los elementos que tienen el primer índice común. Por último, el tercer sumando es el que nos lleva, definitivamente, a la dirección del elemento específico; en efecto, equivale a un desplazamiento relativo al valor del segundo índice.
Este método es extensivo para el cálculo de las posiciones en las que se almacenan los elementos de los "arrays" de cualquier número de dimensiones.

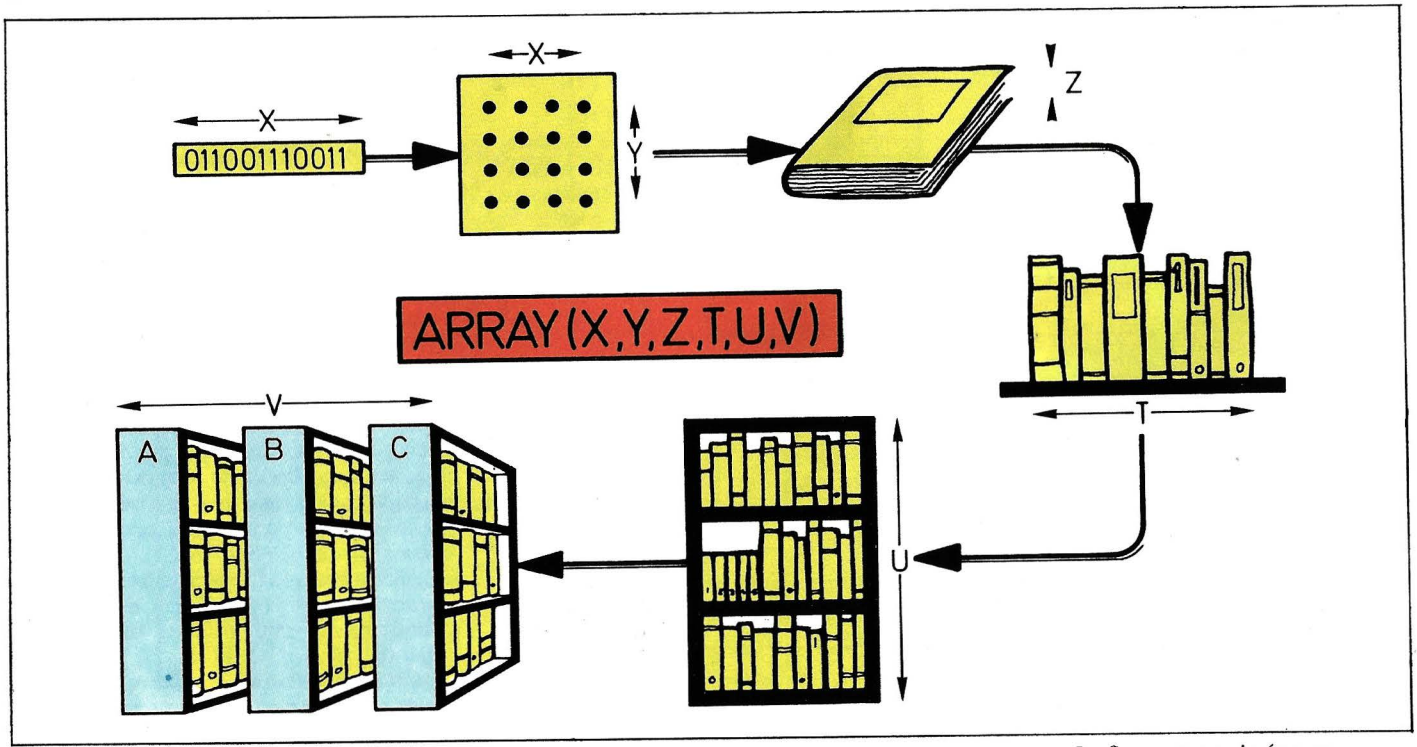


En este caso, la operación consiste en añadir el valor del elemento genérico a la variable SUMA. Al acumularlos todos en SUMA, se habrá obtenido la suma total de los ocho elementos. Para evaluar la calificación media basta tan sólo con dividir el valor de SUMA entre 8, al producirse la salida del bucle (línea 90). El cometido de la línea 50 no es otro que "borrar" el contenido que anteriormente pudiera tener la variable SUMA, para tener la seguridad de que sólo contribuirán a la suma total los elementos de MEDIA.

LA SEGUNDA DIMENSION

Los conjuntos de variables utilizados hasta ahora se denominan "lineales" o de *una dimensión,* ya que sólo poseen un índice. En efecto, los elementos de estos "arrays" pueden ordenarse en una línea, al contar con un único índice de referencia.

Estos conjuntos de variables resultan útiles para el tratamiento de datos con una sola característica diferenciadora (un solo



Los conjuntos de variables o "arrays" de variables suscritas pueden ser de cualquier número de dimensiones. La figura aporta imágenes sugerentes de conjuntos de variables de una a seis dimensiones.

"grado de libertad"). Así, por ejemplo, en el conjunto de notas medias, la característica identificativa era el curso de EGB. En determinadas ocasiones, es preciso diferenciar más de una cualidad de cada dato. Por ejemplo, el conjunto de notas medias de varios alumnos. Ahora, aparte del identificador de curso, habrá que introducir el identificador del alumno. Ya no basta con determinar el curso al que se hace referencia, puesto que cada alumno tendrá un dato distinto de nota media en dicho curso. Este segundo conjunto puede representarse como una tabla, en la que las filas corresponden a los alumnos y las columnas a los cursos. En definitiva, la ordenación se realiza en dos direcciones: se trata, pues, de un conjunto de *dos dimensiones*.

Para acceder a estos nuevos "arrays" o matrices, son necesarios dos índices. Su formulación en BASIC tiene el siguiente aspecto: MEDIA(3,7).

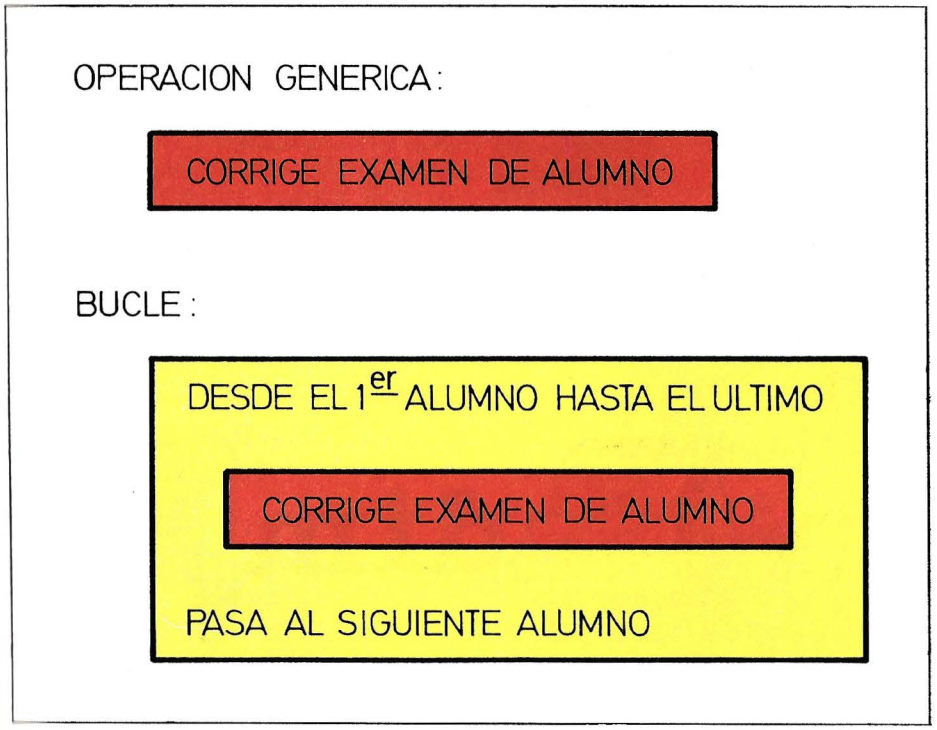
Los dos índices se encuentran encerrados entre paréntesis y separados entre sí por una coma. La forma de trabajo es análoga a la que corresponde a los conjuntos de variables de una sola dimensión. Por ejemplo, para visualizar las notas del alumno número uno, las instrucciones adecuadas serían:

```
120 FOR INDICE=1 TO 8
130 PRINT MEDIA(INDICE,1)
140 NEXT INDICE
■
```

Su similitud con el caso de una sola dimensión es obvia. En realidad, un conjunto de variables de dos dimensiones viene a ser un "conjunto de subconjuntos"; con el primer índice se numeran los elementos de cada subconjunto y con el segundo los subconjuntos.

Los conjuntos de dos dimensiones resultan perfectamente adecuados para almacenar matrices. En tal caso, uno de los índices representa a la fila y el otro a la columna que ocupa cada elemento. La elección del índice que corresponde a la fila o a la columna es totalmente libre, siempre que el resto de las operaciones a realizar sea consecuente con esta elección.

Para tratar recursivamente un "array" de dos dimensiones no basta con un bucle



Para ejecutar una operación con todos los elementos de un conjunto de variables, es preciso definir la operación sobre un elemento genérico y encerrarla dentro de un bucle reiterativo que afecte, sucesivamente, a todos los elementos.

Ejemplo de "array" de dos dimensiones

El siguiente programa ilustra el tratamiento de una variable suscrita de dos dimensiones. Su cometido se reduce a definir una matriz de 3×3 elementos (línea 5), solicitar la introducción de los nueve datos y, finalmente, mostrarlos en la pantalla. Tal y como se ha descrito en el texto, la manipulación recursiva de un conjunto de variables de doble dimensión (dos índices), exige la presencia de dos bucles anidados. En el caso que nos ocupa, los valores iniciales de las instrucciones FOR especifican el valor 1 como primer índice.

```
 5 DIM A(3,3)
10 FOR I=1 TO 3
20 FOR J=1 TO 3
30 PRINT "INTRODUCE EL ELEMENTO
   A(";I;",";J;")"
40 INPUT A(I,J)
50 NEXT J
60 NEXT I
100 FOR I=1 TO 3
110 FOR J=1 TO 3
120 PRINT A(I,J),
130 NEXT J
140 PRINT
150 NEXT I
160 END
```

La ejecución del programa se traduce en la siguiente secuencia en la pantalla del ordenador:

```
INTRODUCE EL ELEMENTO A(1,1)
1
INTRODUCE EL ELEMENTO A(1,2)
2
INTRODUCE EL ELEMENTO A(1,3)
3
INTRODUCE EL ELEMENTO A(2,1)
4
INTRODUCE EL ELEMENTO A(2,2)
5
INTRODUCE EL ELEMENTO A(2,3)
6
INTRODUCE EL ELEMENTO A(3,1)
7
INTRODUCE EL ELEMENTO A(3,2)
8
INTRODUCE EL ELEMENTO A(3,3)
9

 1 2 3
 4 5 6
 7 8 9
```

FOR/NEXT, sino que hay que construir un bucle por cada una de las dimensiones. Uno de ellos recorrerá los elementos de cada fila (columna a columna) y el otro irá recorriendo las sucesivas filas.
Pasemos al terreno práctico, suponiendo que el primer índice representa la fila y el segundo al elemento específico que ocupa una determinada posición dentro de la misma. La operación a realizar coincidirá con una asignación de valor a los elementos de la matriz. La siguiente línea de instrucción resulta apropiada para tal cometido:

```
40 LET MATRIZ(FILA,ELEMENTO)=0
```

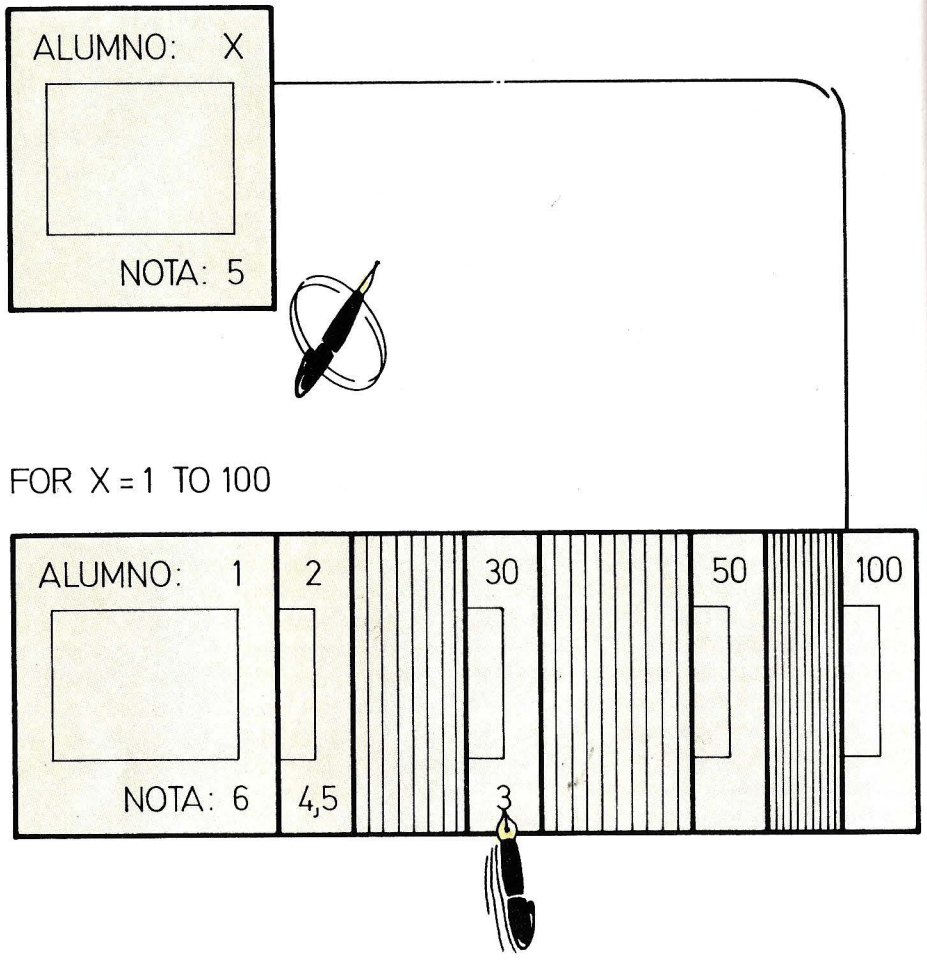
Desde luego, esta línea representa a una asignación genérica. Ahora, es necesario crear el bucle que recorra todos los elementos de una determinada fila. En el ejemplo, supondremos que cada fila consta de diez elementos:

```
30 FOR ELEMENTO=1 TO 10
40 LET MATRIZ(FILA,ELEMENTO)=0
50 NEXT ELEMENTO
```

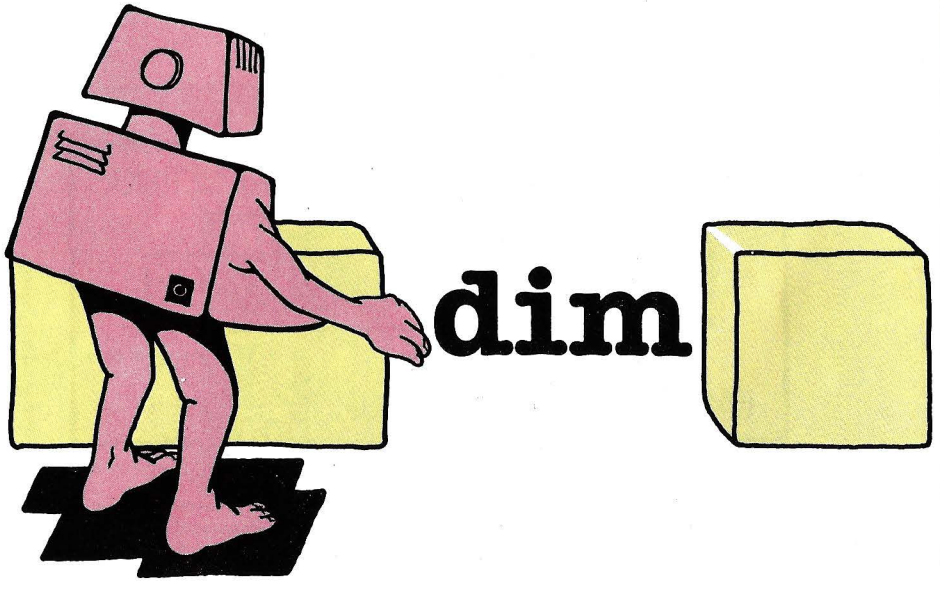
Tres instrucciones BASIC son suficientes para asignar un valor (el cero en nuestro caso) a todos los elementos de una fila.
A continuación trataremos a estas tres líneas de programa como un bloque unitario, para extender la asignación a las seis filas de elementos que consideramos en nuestra matriz ejemplo. Para ello, crearemos un nuevo bucle externo que recorra todas las filas:

```
30 FOR FILA=1 TO 6
30    FOR ELEMENTO=1 TO 10
40       LET MATRIZ(FILA,ELEMENTO)=0
50    NEXT ELEMENTO
70 NEXT FILA
■
```

Ambos bucles rodean a la línea de instrucción genérica que se ocupa de ordenar la asignación (línea 40). Se observa que los dos bucles aparecen "anidados". Cabe recordar en este punto que, cuando se crean bucles anidados, es indispensable cerrar con la palabra clave NEXT los bucles interiores antes que los exteriores.



La figura muestra un ejemplo elocuente de operación que afecta a todos los elementos de un conjunto de variables. La operación a realizar (otorgar la calificación media a cada alumno), puede definirse a modo de operación genérica dentro de un bucle; éste se ocupará de que la operación se realice sobre las hojas de calificación de los cien alumnos.



CONJUNTOS DE MULTIPLES DIMENSIONES

Si un conjunto de una dimensión equivalía a puntos situados en línea, y uno de dos dimensiones a una tabla o retícula (algo semejante a un tablero de ajedrez), la tercera dimensión cabe imaginarla como un conjunto ordenado según las tres direcciones espaciales. En este caso, además de filas y columnas cabe pensar en una tercera referencia, por ejemplo, en el "nivel de profundidad". Los "arrays" de tres dimensiones se tratan igual que los de dos o una dimensión. La única diferencia radica en que ahora serán tres los índices y los bucles anidados para su tratamiento recursivo.

Volviendo al ejemplo inicial, si la primera dimensión indicaba las notas de un alumno y la segunda reflejaba a los distintos alumnos, cabe la posibilidad adicional de agrupar a los alumnos por colegios. En tal caso, el tercer índice será el indicativo de los diversos colegios considerados. Con esta estructura tridimensional, será posible acceder a la nota del curso X, concretamente del alumno Y, que cursa sus estudios en el centro Z.

Pero las dimensiones no acaban aquí. En BASIC se puede saltar a la cuarta dimensión, a la quinta, a la sexta, ...

Por cada nueva dimensión la cosa se complica un poco más. Es necesario incluir un nuevo índice y, para realizar una operación con todos los elementos del conjunto, es preciso anidar un nuevo bucle FOR/NEXT. Afortunadamente, el trabajo habitual se reduce a "arrays" de a lo sumo tres o cuatro dimensiones.

DANDO TAMAÑO A LOS CONJUNTOS

El tamaño de los conjuntos de variables con los que se puede trabajar en un ordenador no es ilimitado. Como es lógico, depende de la capacidad de memoria del equipo.

TABLA DE CONVERSION

ORDENADOR	DIM		OPTION BASE
	DIM <var.>	DIM <lista>	OPTION BASE <ind.>
APPLE II (APPLESOFT)	DIM <var.>	DIM <lista>	—
APRICOT (M-BASIC)	DIM <var.>	DIM <lista>	OPTION BASE <ind.>
ATARI	DIM <var.>	DIM <lista>	—
CBM 64	DIM <var.>	DIM <lista>	—
DRAGON	DIM <var.>	DIM <lista>	—
EQUIPOS MSX	DIM <var.>	DIM <lista>	—
HP-150	DIM <var.>	DIM <lista>	OPTION BASE <ind.>
IBM PC	DIM <var.>	DIM <lista>	OPTION BASE <ind.>
MPF	DIM <var.>	—	—
NCR DM-V (MS-BASIC)	DIM <var.>	DIM <lista>	OPTION BASE <ind.>
NEW BRAIN	DIM <var.>	DIM <lista>	OPTION BASE <ind.>
ORIC	DIM <var.>	—	—
SHARP MZ-700 (MZ-BASIC)	DIM <var.>	DIM <lista>	—
SINCLAIR QL	DIM <var.>	DIM <lista>	—
SPECTRAVIDEO	DIM <var.>	DIM <lista>	—
ZX-SPECTRUM	DIM <var.>	—	—

<var.>: Nombre de variable de conjunto o "array". <lista>: Lista de variables de conjunto. <ind.>: Indice inicial (0 ó 1).

FORMULACIONES DE LOS COMANDOS

DIM <var.]: Reserva espacio en memoria para una variable de conjunto. DIM [lista]: Reserva espacio para varios "arrays" por medio de un solo comando DIM. OPTION BASE: Fija el índice inicial del "array".

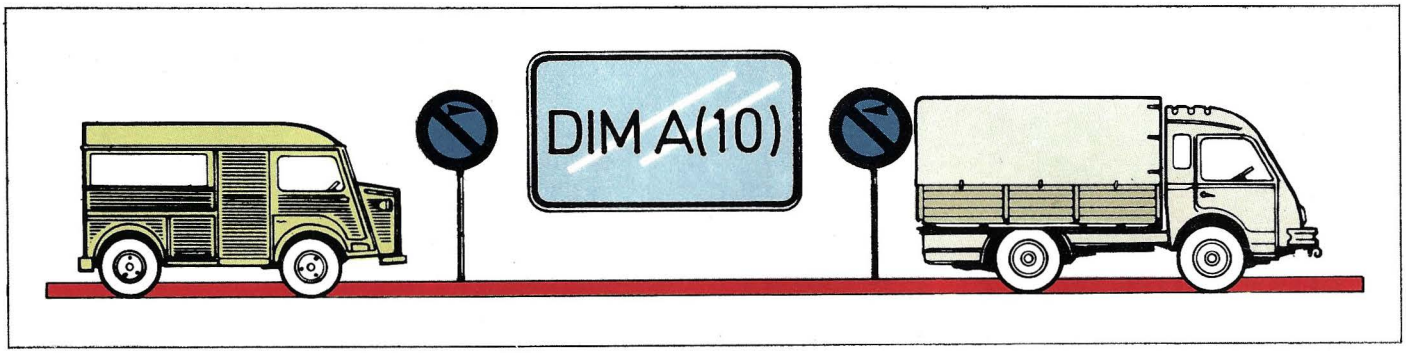
Cuando se define una variable, el ordenador se encarga de "reservarle sitio" automáticamente. Cuando se trata de conjuntos o "arrays" el proceso es más complejo, puesto que exigen un mayor espacio de memoria. Además, el espacio ne-

DIM

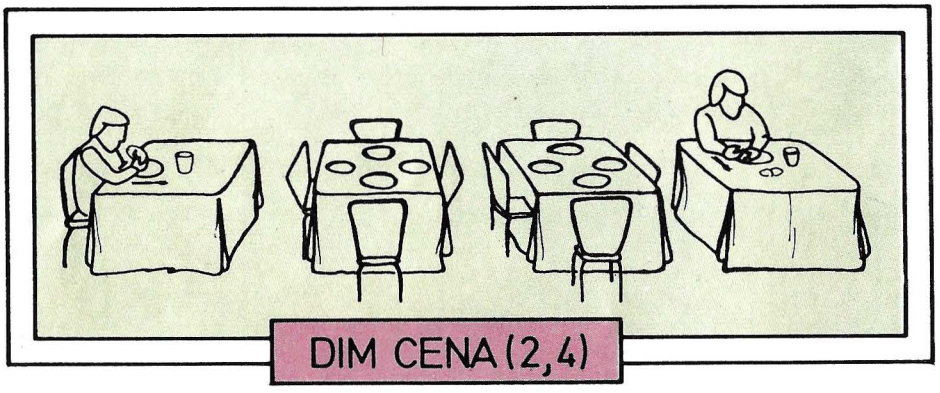
Dimensiona el tamaño máximo de una o varias variables de conjunto.

Formato: (Número de línea) DIM <lista de variables de conjunto>

Ejemplos: 10 DIM A(100)
50 DIM OROS(10),COPAS(10),BAR(4,10)



La función del comando DIM es la de reservar espacio en memoria para almacenar a todos los elementos que constituyen el conjunto de variables o "array".



Al formular una instrucción DIM, hay que especificar los valores máximos que pueden adoptar los índices de la variable suscrita. En el ejemplo de la figura, la instrucción DIM reservará dos mesas para cuatro comensales cada una.

cesario va a depender del número máximo de elementos del conjunto. Para evitar que en un momento dado no quede suficiente memoria para almacenar un "array", es necesario reservar previamente el espacio preciso.

Esta "reserva" de espacio se realiza en BASIC mediante una instrucción llamada DIM (de DIMensionar). En el argumento de DIM se especifica el nombre del "array", seguido por la longitud máxima de sus dimensiones. Para reservar el espacio necesario para la tabla de calificaciones de los 8 cursos de EGB, cada uno con 10 alumnos, habrá que utilizar la instrucción:

DIM MEDIA(8,10)

De esta forma el ordenador "toma conciencia" de la magnitud del conjunto de variables.

En términos generales, el formato de DIM es el siguiente:

(Número de línea) DIM <lista de variables del conjunto>

Habitualmente, se puede dimensionar más de un "array" en cada instrucción DIM. Esto se consigue separando los conjuntos a definir por medio de comas. Por ejemplo:

DIM GASTOS(12),INGRESOS(12),
TOTALES(3,12)

El dimensionamiento de los "arrays" o conjuntos de variables suscritas, suele realizarse al principio del programa. De esta forma, no cabe el riesgo de operar con un conjunto de variables que no haya sido definido previamente. En la mayoría de los ordenadores, existe un método de dimensionado automático para "arrays" pequeños. Si no se fija la longitud, el traductor BASIC supone que se trata de un conjunto de 10 elementos. Así pues, la línea de instrucción 10 DIM A(10), B(10), C(10) puede omitirse si el ordenador posee esta característica.

EL ELEMENTO CERO

Hemos comentado que los índices definen una ordenación: desde el 1 hasta el valor máximo que tomen. En algunos casos —sobre todo en aplicaciones matemáticas—, resulta interesante disponer del índice cero. Al respecto cabe señalar que algunos dialectos BASIC no admiten la presencia del índice cero, mientras que otros dialectos lo incorporan a cualquier "array" definido. Un tercer grupo de traductores BASIC permite elegir la opción adecuada.

Los ordenadores que permiten elegir el índice del primer elemento poseen, al efecto, el comando OPTION BASE. En el argumento de OPTION BASE se especifica cuál debe ser el índice inferior: 1 ó 0.

OPTION BASE

Permite elegir el valor mínimo del índice entre uno y cero.

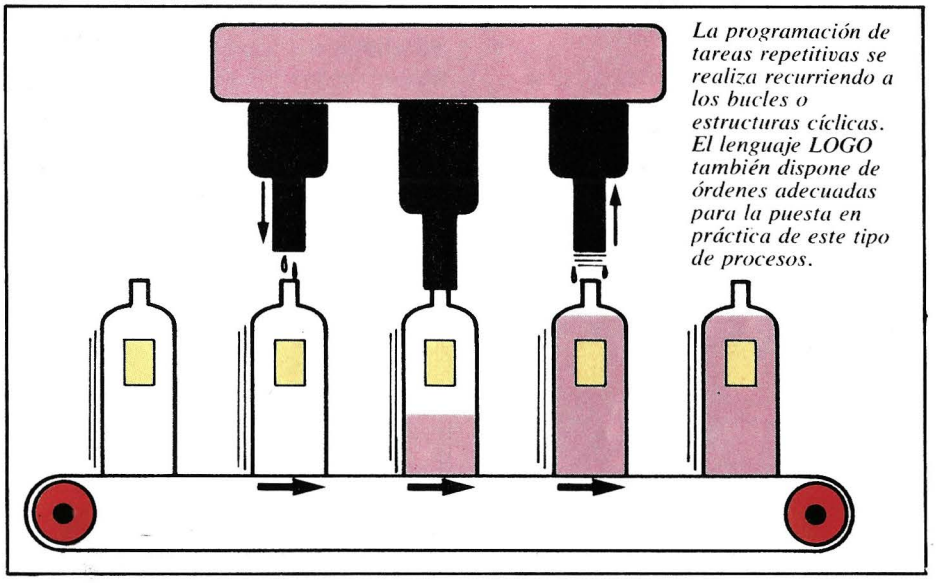
Formato: (Número de línea) OPTION BASE <índice>

Ejemplos: 10 OPTION BASE 0
12 OPTION BASE 1

Logo (11)

Programación de bucles y detección de colisiones

En los capítulos anteriores se ha hablado de los dos aspectos distintos del LOGO: el tratamiento de gráficos y el uso de palabras y listas. Dentro de estos temas, a menudo se ha hecho referencia a posibilidades más complejas; algunas de ellas son las que se comentan a continuación.



La programación de tareas repetitivas se realiza recurriendo a los bucles o estructuras cíclicas. El lenguaje LOGO también dispone de órdenes adecuadas para la puesta en práctica de este tipo de procesos.

BUCLES

Un hecho evidente es que el ordenador ha sido creado para descargar al usuario de las tareas rutinarias. Este concepto es ampliamente contemplado en la programación de bucles.
En el siguiente ejemplo se utiliza a la tortuga para dibujar un cuadrado:

```
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
```

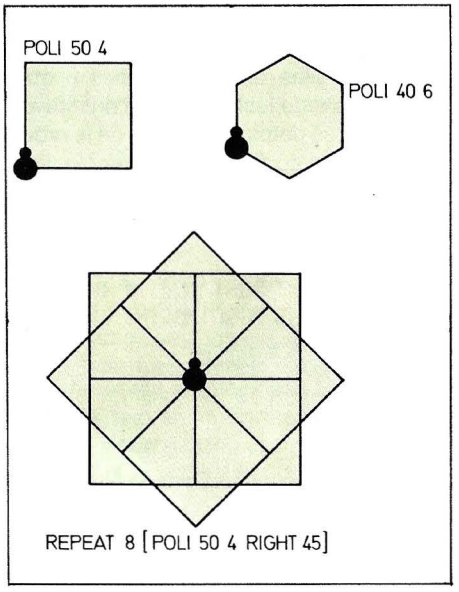
Se observa que las instrucciones FORWARD 50 y RIGHT 90 se repiten en cuatro ocasiones. El usuario se ve obligado a teclear cuatro veces las mismas órdenes.
Este hecho queda subsanado con el uso de REPEAT.
El comando REPEAT es la primera orden especializada en la creación de bucles. Su presencia simplifica el ejemplo anterior a su mínima expresión:

```
REPEAT 4 [FORWARD 50 RIGHT 90]
```

En efecto, una sola línea de instrucción suple a las ocho que constituían la primera alternativa para dibujar el cuadrado. El comando REPEAT admite dos datos de entrada. El primero ha de ser un dato numérico que especifica el número de veces que debe repetirse una acción. Dicha acción se introduce como segundo dato de entrada. Este último puede coincidir con una lista de instrucciones.
El procedimiento que sigue, POLI, hace uso de REPEAT para dibujar un polígono:

```
TO POLI :LONG :NUM
REPEAT :NUM [FORWARD :LONG
RIGHT (360/:NUM)]
END
```

El dato LONG define la longitud del lado, mientras que NUM indica el número de lados del polígono. El cuadrado anterior puede también confeccionarse por medio de POLI 50 4.



*El procedimiento **POLI**, definido en el texto, incluye un bucle construido con la colaboración de la orden **REPEAT**. La figura ilustra algunos ejemplos prácticos basados en dicho procedimiento.*

BIFURCACIONES

En algún caso, puede ser necesario que dentro de un procedimiento se realicen distintas acciones, dependiendo de ciertas condiciones impuestas. Por ejemplo, se puede definir un procedimiento que calcule raíces cuadradas mediante el operador SQRT:

```
TO RAIZ :N
PRINT SQRT :N
END
```

Este procedimiento dará error si el número introducido es negativo (no es posible hallar la raíz cuadrada de un número negativo). Para evitarlo, puede ordenarse que el cálculo de la raíz cuadrada se efectúe exclusivamente si el número N es positivo.

```
TO RAIZ :N
IF :N > 0 [PRINT SQRT :N]
END
```

En este segundo caso sólo se calcula la raíz si el número es mayor que cero (N > 0). La orden IF sirve para "bifurcar" la ejecución. Esta orden admite tres datos de entrada. El primero corresponde a una condición, esto es: a un enunciado lógico cuya evaluación dará como resultado un dato de tipo lógico. Dependiendo de si el valor resultante es TRUE o FALSE, se ejecutará una lista de instrucciones u otra. Ambas listas de instrucciones constituyen los restantes datos de entrada de la orden IF. Siguiendo con el mismo ejemplo, veamos cuál es el nuevo procedimiento, ligeramente más complejo:

```
TO RAIZ :N
IF :N > 0 [PRINT SQRT :N] [PRINT
[SOLO NUMEROS POSITIVOS]]
END
```

En el primer caso sólo se utilizó una lista de instrucciones, mientras que ahora se contempian ambas posibilidades. Si :N no es mayor que 0, se ejecutará la segunda lista de instrucciones.
El enunciado de la condición ha de producir un dato de tipo lógico. En consecuencia, este enunciado puede incluir todo tipo de operadores lógicos. Aquí es donde los operadores AND, OR, NOT adquieren una importancia especial. Suponga, por ejemplo, que es preciso que N sea mayor o igual que 0. Tal condición queda plasmada en la expresión siguiente:

```
IF OR :N    0 :N =0 ...
```

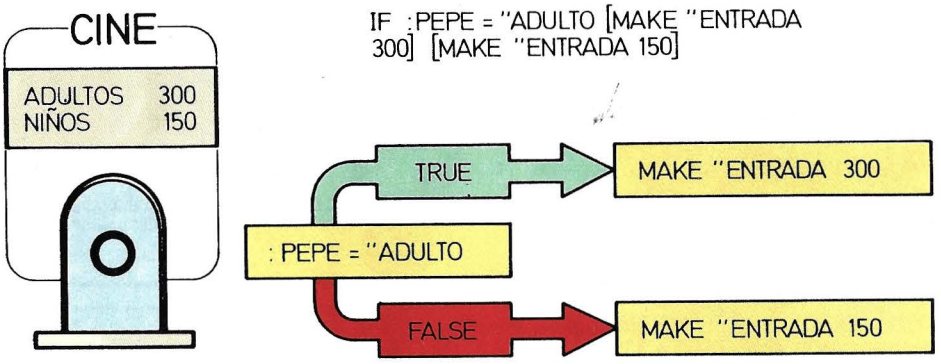
De igual forma, también está permitido el uso de otros operadores como LISTP, WORDP, SHOWNP, etc., para bifurcar la ejecución. Estos últimos permiten el tratamiento diferenciado de los distintos tipos de datos.

```
TO RAIZ :N
IF NOT NUMBERP :N [PRINT [HAY QUE
APORTAR UN NUMERO]] [IF OR :N > 0 :N =0
[PRINT SQRT :N] [PRINT [SOLO NUMEROS
POSITIVOS]]]
END
```

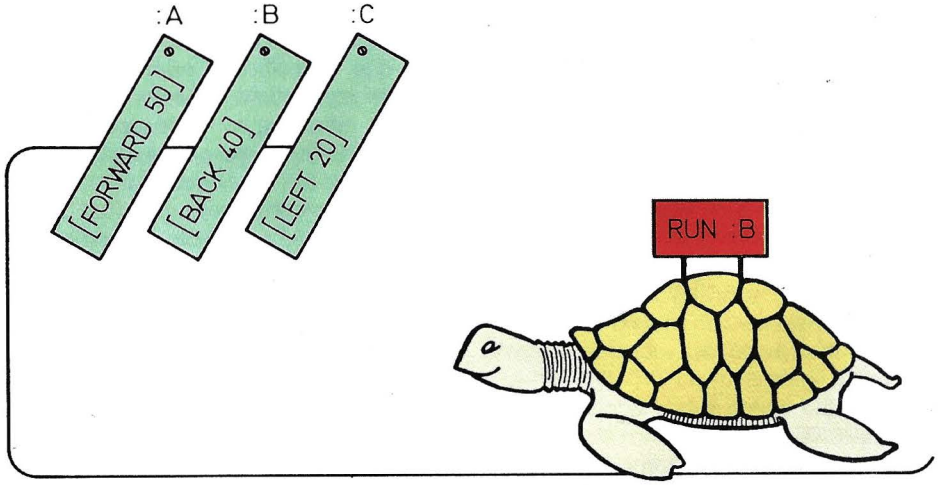
Ahora puede verse la utilidad de operadores como NUMBERP. La ejecución de este último procedimiento aparece en la siguiente pantalla.

```
RAIZ "CUATRO
HAY QUE APORTAR UN NUMERO
RAIZ-4
SOLO NUMEROS POSITIVOS
RAIZ 4
2
RAIZ 0
0
```

La instrucción IF actuará como comando u operador según la naturaleza de las órdenes contenidas en las listas. En los ejemplos precedentes realizaba una acción propia de un comando, desencadenada por la presencia de PRINT. Veamos un nuevo ejemplo en el que actúa a modo de comando:



IF es una orden LOGO adecuada para bifurcar la ejecución de un programa, de acuerdo al cumplimiento de la condición impuesta. En el ejemplo, la condición "ADULTO qué dato se asignará a la variable "ENTRADA.



RUN es una nueva orden LOGO especializada en la ejecución de una lista de instrucciones. Esta lista, incluida como dato de entrada de RUN, puede verse reemplazada por la correspondiente variable representativa de la lista a ejecutar.

```
TO RAIZOP :N
IF NOT NUMBERP :N [OUTPUT [HA DE
APORTAR UN NUMERO]] [IF OR :N > 0 :N =0
[OUTPUT SQRT :N][OUTPUT [SOLO
NUMEROS POSITIVOS]]]
END
```

Su ejecución será:

```
PRINT RAIZOP "CUATRO
HA DE DARSE UN NUMERO
PRINT RAIZOP -4
SOLO NUMEROS POSITIVOS
RAIZOP 4
YOU DON'T SAY WHAT TO
DO WITH 2
PRINT RAIZOP 4
2
PRINT RAIZOP 0
0
```

Al actuar en modo operador, IF entrega un dato de salida. Este dato debe constituir la entrada a un comando; en caso contrario, el LOGO escribirá en la pantalla un mensaje de error.

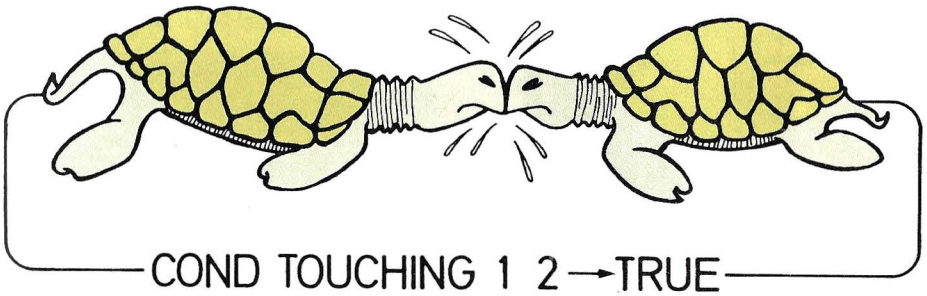
EJECUCION DE UNA LISTA

Tal como se ha estudiado, las órdenes REPEAT e IF admiten una lista de órdenes como dato de entrada. En ambos casos, la lista incluye las instrucciones a ejecutar.
Cabe observar que el contenido de la lista no tiene por qué ser fijo. Es posible hacer uso de una variable cuyo contenido coincida con la lista específica de órdenes. El siguiente procedimiento ilustra lo indicado:

```
TO NUMBER :N
MAKE "POSI [PRINT "POSITIVO]
MAKE "NEGA [PRINT "NEGATIVO]
IF :N < 0 : NEGA : POSI
END
```

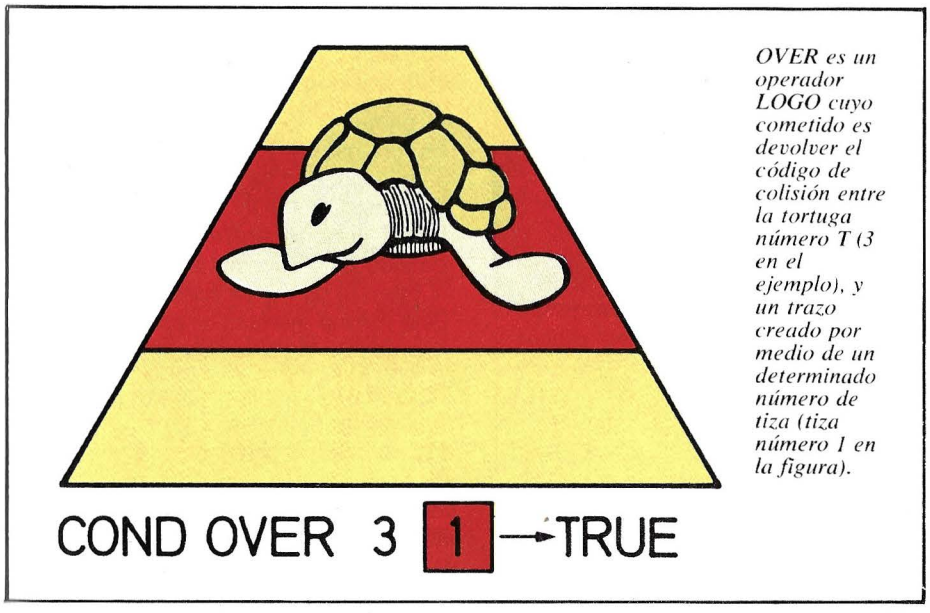
El procedimiento utiliza dos variables (:NEGA y :POSI) para almacenar la lista de instrucciones.
Existe otro comando LOGO adecuado para ejecutar una lista de instrucciones: RUN.
Su dato de entrada debe coincidir con la lista a ejecutar. Normalmente, la lista se verá reemplazada por una variable que adecuará su contenido para que se ejecute una y otra lista de órdenes LOGO.

```
TO NUMBER :N
MAKE "POSI [PRINT "POSITIVO]
MAKE "NEGA [PRINT "NEGATIVO]
IF :N<0[MAKE "L :NEGA]
[MAKE "L :POSI]
RUN :L
END
```



La propia tortuga del LOGO puede introducir criterios para bifurcar la ejecución; por ejemplo, a través de la orden COND, capaz de detectar la ocurrencia o no del tipo de colisión definida.

COLISIONES



OVER es un operador LOGO cuyo cometido es devolver el código de colisión entre la tortuga número T (3 en el ejemplo), y un trazo creado por medio de un determinado número de tiza (tiza número 1 en la figura).

La tortuga también puede introducir criterios para bifurcar la ejecución. Al margen de las funciones lógicas ya estudiadas, existe una nueva función adecuada para detectar colisiones. El operador COND devuelve un valor lógico indicando si se produce la colisión definida.
Las colisiones pueden ocurrir al coincidir una tortuga con una determinada porción de la pantalla, o al entrar en contacto dos tortugas. Las distintas combinaciones posibles se codifican por medio de un número. Este se añade a continuación del operador COND para que este detecte si se produce tal evento. Los códigos de colisión dependen de cada dialecto

TABLA DE ORDENES-LOGO

Instrucción	Cometido	Operador/ Comando
REPEAT <n><lista>	Repite n veces las órdenes indicadas en la lista	Comando
IF<cond><lista1> <lista2>	Si la condición es cierta se ejecuta la lista1, en caso contrario se ejecuta la lista2.	Depende de la lista
COND <n>	Devuelve el valor TRUE si la colisión número n se cumple	Operador
TOUCHING <T1> <T2>	Devuelve el número identificativo de la colisión entre las tortugas número T1 y T2	Operador
OVER<Tor><Tiza>	Devuelve el código de colisión entre la tortuga número T y la tiza cuyo número se indica	Operador
RUN<lista>	Ejecuta las órdenes indicadas en la lista	Depende de la lista
WHEN<n><lista>	Ejecuta la lista en el mismo instante en que se produzca la colisión número n	Depende de la lista

NOTA: <n> Dato numérico (número de veces o número de colisión). <lista>, <lista1> y <lista2>: Lista de instrucciones. <Tor>, <T1> y <T2>: Números de tortuga. <Tiza>: Número de tiza. <cond>: Condición que genera un número de tipo lógico.

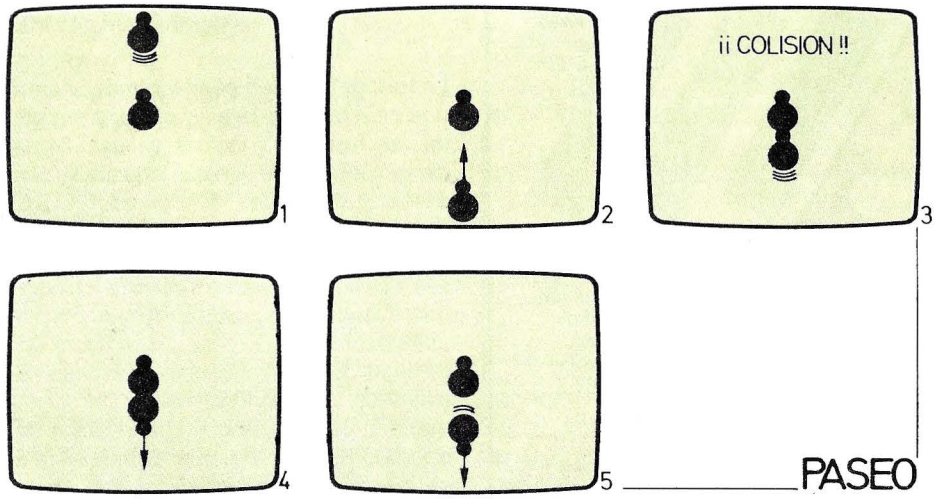
LOGO; en cualquier caso, es posible evitar su empleo gracias a otros dos operadores.

TOUCHING es un operador que devuelve el código de la colisión entre dos tortugas. Su argumento incluye los números identificativos de las tortugas deseadas. Así pues, para detectar una colisión entre las tortugas 1 y 2 basta con explorar el dato de salida de COND TOUCHING 1 2.

Para identificar la coincidencia de posición de una tortuga y un trazo en la pantalla se hace uso de OVER. Así, por ejemplo, la siguiente lista de órdenes mostrará el valor lógico de la colisión de la tortuga 0 con un trazo creado en la pantalla con la tiza número 2:

```
PRINT COND OVER 0 2
```

El dato de salida de COND puede identificarse dentro de una orden IF, para bifurcar hacia acciones distintas. De esta forma, se puede ejecutar una secuencia de instrucciones específica cuando se produzca una colisión.



Secuencia de ejecución del procedimiento PASEO.

DETECCION INSTANTANEA

El problema que subyace en el uso conjunto de IF y COND, deriva de la necesidad de ejecutar dichas órdenes repetidamente. Si la colisión se produce antes o después de chequear la condición, su efecto no se verá reflejado al ajecutar el procedimiento. En definitiva: la colisión sólo se detecta al ejecutarse la cláusula COND. Se puede utilizar otro método para detectar colisiones "en el acto". El comando implicado es WHEN. Los dos parámetros de entrada de WHEN son el código de colisión y una lista de instrucciones. Las instrucciones de la lista se ejecutarán *inmediatamente,* en cuanto se produzca la colisión.

Veamos un ejemplo ilustrativo de ambos métodos de detección.

```
TO PASEO
TELL 0 1
ST
CS
PU
ASK 0 [FORWARD 50 SETSP 50]
COLISION
END
```

```
TO COLISION
IF COND TOUCHING 0 1 [ASK 0 [RIGHT
180 FORWARD 10]]
COLISION
END
```

El primer procedimiento PASEO llama a las tortugas 0 y 1. La número 0 se moverá a una velocidad de 10. A su vez, el procedimiento COLISION se encarga de detectar el choque entre ambas tortugas. Cuando esto sucede, la tortuga móvil (la número 0) da media vuelta.

Este mismo ejemplo es programable con la ayuda de WHEN:

```
TO PASE01
TELL 0 1
ST
CS
PU
ASK 0 [FORWARD 50 SETSP 50]
WHEN TOUCHING 0 1 [ASK 0 [RIGHT 180]]
END
```

CP/M-86 y derivados del CP/M

La llegada de los microprocesadores de 16 bits

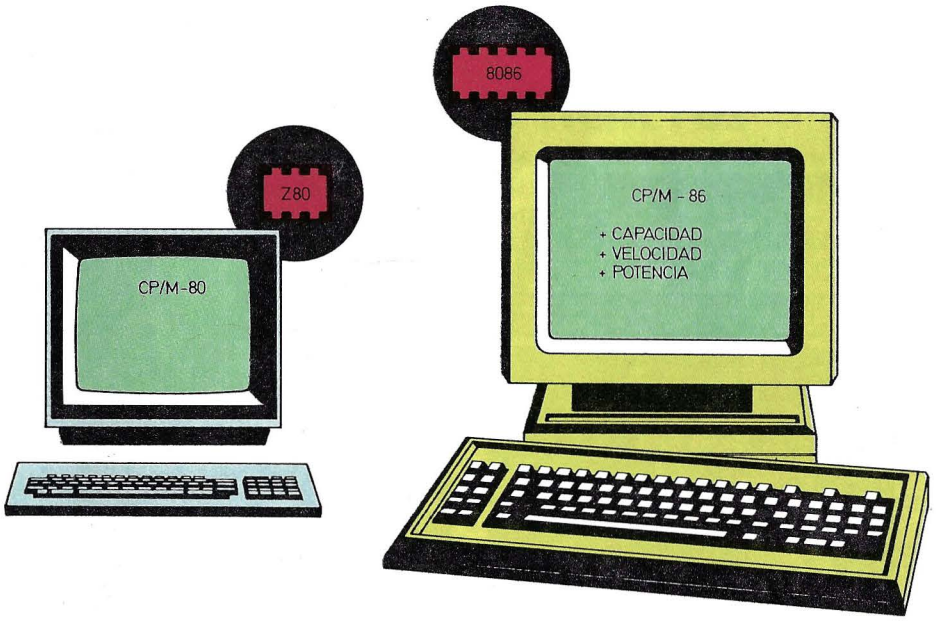
Las primeras versiones del sistema operativo CP/M se desarrollaron para el trabajo con sistemas basados en un microprocesador de 8 bits; tal es el caso de los microprocesadores 8080 y 8085 de la firma Intel. Un nuevo paso, también en el terreno de los 8 bits, llegó con el CP/M 80: una versión adecuada para microordenadores cuya CPU está regida por el microprocesador Z-80.

A pesar de la indudable potencia de las referidas versiones del CP/M, todas ellas tropiezan con la limitación impuesta por la naturaleza del microprocesador. El problema surge a la hora de ejecutar programas de elevada complejidad, que exigen un gran volumen de memoria central para su almacenamiento. En tal caso, y a pesar de la eficaz colaboración gestora del sistema operativo, la ejecución se ve comprometida por los 64 Kbytes que constituyen el espacio de memoria direccionable por un microprocesador de 8 bits. ¿Cómo es posible proceder a la ejecución de un programa que ocupa más de los 64 Kbytes de memoria central disponibles en el mejor de los casos?

El CP/M puede aportar su capacidad organizativa y de control hasta el punto de brindar una primera solución. El método consiste en fraccionar el programa en módulos que puedan ejecutarse por separado, de tal forma que uno de ellos resida siempre en la memoria central del ordenador. Este es el denominado *módulo raíz,* cuya función es ir "llamando" a los demás módulos a medida que lo exija el desarrollo del programa. Los diferentes módulos irán trasladándose desde el dispositivo de almacenamiento secundario (un disco flexible, por ejemplo) a la memoria primaria del sistema, de tal forma que el volumen ocupado por el módulo raíz y los diferentes módulos que lo acompañen nunca exceda el límite de capacidad de la memoria primaria.

Aunque este método se manifiesta eficaz en algunos casos, su puesta en práctica puede conducir a una desmembración tal del programa que su ejecución resulte inviable, debido a las constantes entradas y salidas de módulos de la memoria interna.

La llegada de los microprocesadores de 16 bits trajo la solución. Paralelamente, Digital Research, creadora del CP/M, desarrolló nuevas versiones de este sistema operativo adecuadas para los microprocesadores de 16 bits 8086 y 8088; tal es el caso del CP/M-86. Quedaba ya resuelto el problema antes señalado. La nueva era de los 16 bits suponía una mejora espectacular en la capacidad de memoria primaria, al permitir el direccionamiento de más de un millón de posiciones de memoria.



La irrupción de los microprocesadores de 16 bits en el ámbito de la microinformática, ha acercado la potencia y capacidad de los ordenadores personales a cotas hasta hace poco exclusivas de los grandes equipos.
La evolución circuital que ello ha supuesto, se ha visto apoyada por el desarrollo de nuevos sistemas operativos más potentes y capaces.

ARQUITECTURA DEL CP/M-86

La estructura interna del CP/M-86 es muy parecida a la propia del CP/M-80. No hay que olvidar que uno de los objetivos de Digital Research fue desarrollar un sis-

tema operativo que presentara la mayor compatibilidad posible con su antecesor. Por supuesto, y debido a la distancia que media entre ambas versiones, hay ciertas particularidades y ampliaciones exclusivas del CP/M-86.

La mayor parte de las utilidades presentes en el CP/M-80 se conservan tal y como existían anteriormente, salvo la distribución de algunas zonas de memoria y la ubicación de las extensiones. Los módulos que dan cuerpo al CP/M-86 siguen siendo los que rigen la estructura del CP/M-80.

— CCP: procesador de comandos de consola.

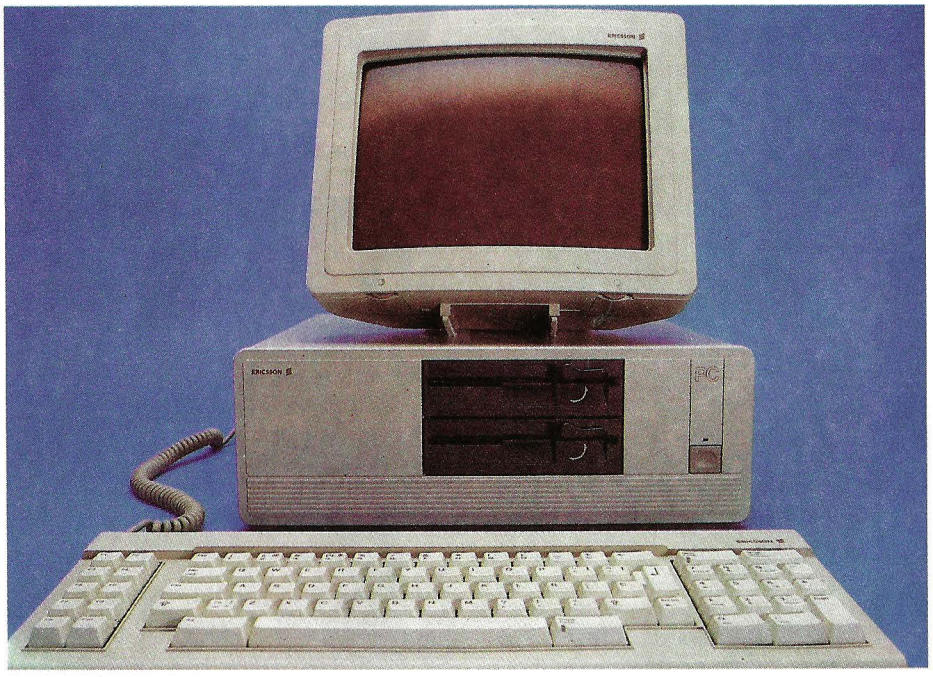
— BDOS: sistema operativo básico de disco.

— BIOS: sistema básico de entrada/salida.

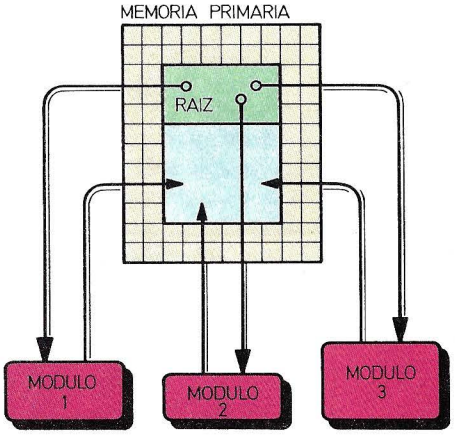
El nuevo módulo CCP también incorpora comandos residentes y transitorios. Los residentes no presentan ninguna novedad que los distancie de sus homólogos en el CP/M-80. Si bien, por lo que respecta a los comandos transitorios residentes en disco, hay que señalar que ahora están alojados en ficheros cuyo tipo viene determinado por las tres letras "CMD"; en lugar de "COM", como era característico en el CP/M-80. Este cambio en la nomenclatura del tipo de fichero que contiene a los comandos transitorios tiene una razón de peso. Como las versiones CP/M-80 y CP/M-86 utilizan el mismo método de almacenamiento en disco, puede darse el caso de que se almacenen en un mismo disco comandos transitorios pertenecientes a cada uno de los dos sistemas. Para evitar posibles confusiones en la carga y ejecución posterior, Digital Research decidió modificar el tipo de fichero en el que residen los comandos transitorios propios de la versión CP/M-86.

La novedad con respecto al módulo BDOS del CP/M-80 consiste en que su llamada no se realiza tras cargar con el número de la función deseada el registro interno C del microprocesador, ejecutándose a continuación una llamada a la posición de memoria 0005 (hexadecimal). En su lugar, los programas que llaman al BDOS hacen uso de un juego de 224 interrupciones de software; éstas entran en escena después de que el registro interno CL esté cargado con el valor adecuado a la función a activar. Por otra parte, las posiciones de memoria 0006 y 0007, empleadas por el módulo BDOS del CP/M-80, no se utilizan en la versión de 16 bits. Dichas posiciones permanecen desocupadas en previsión de que sea preciso ejecutar programas que procedan del CP/M-80 y utilicen las referidas posiciones.

La residencia física del sistema operativo CP/M-86 no se localiza en las dos primeras pistas del disco, como ocurría con el CP/M-80. Al utilizar un microprocesador de 16 bits y no tener una necesidad acuciante de memoria, el sistema operativo puede permitirse el lujo de crecer incorporando nuevas funciones. Con ello, el tamaño del sistema operativo CP/M-86 no permite que éste resida en dos pistas del disco, sino que precisa un fichero especial, denominado CPM.SYS, en el que almacenarse. No obstante, las dos primeras pistas no quedan desocupadas, ya que en ellas se ubica el cargador de arranque en frío ("Cold Start Loader"). Este resulta al concatenar los cargadores LDBIOS. H86, LDBDOS. H86 y LDCPM. H86 y generar un fichero cargador de tipo CMD, cuya misión es la de cargar el fichero CPM.SYS desde el disco a la memoria del sistema.



La entrada en escena de los microprocesadores de 16 bits, ha dado lugar al desarrollo de nuevos sistemas operativos orientados a equipos basados en un microprocesador de esta naturaleza. Uno de los más relevantes es el CP/M-86, creado para equipos basados en los microprocesadores 8086 y 8088.



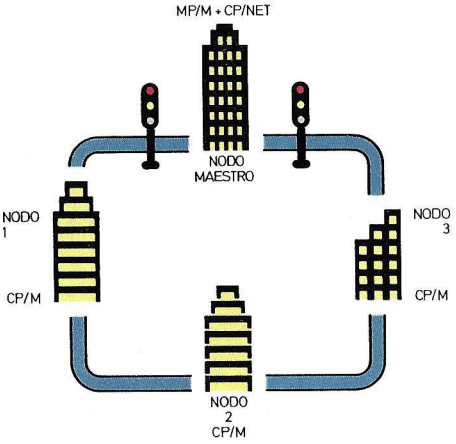
La partición de los programas en distintos módulos puede sortear, en parte, el problema de la reducida capacidad de memoria central de los equipos de 8 bits. En cualquier caso, ésta es una solución marginal no siempre posible en la práctica.

DERIVADOS DEL CP/M

La gran cantidad de software existente para el CP/M ha dado lugar a que bajo su nombre hayan ido apareciendo productos derivados. Productos basados en la misma filosofía que inspira al CP/M y cuyo modo de funcionamiento es práctica-

mente el mismo, salvo en parcelas muy puntuales en donde tratan de mejorar las características de su antecesor.
No todos estos productos alcanzan un nivel satisfactorio de compatibilidad CP/M; algunos de ellos utilizan extensiones ajenas al CP/M original, o almacenan la información en ficheros de distinto formato. En consecuencia, hay que tener un cuidado especial con el software ya existente al pensar en un cambio de sistema operativo, aunque el nuevo sistema esté etiquetado como compatible o "casi-compatible".
Digital Research no ha quedado rezagada en la carrera por la renovación y mejora de los sistemas operativos; son muy diversas las versiones que ha puesto en el mercado de sus CP/M-80, CP/M-86, MP/M-80, MP/M-86, CP/M Concurrente y CP/NET.
Otros fabricantes han lanzado sistemas operativos descendientes directos del CP/M. Sistemas operativos que responden a la necesidad de revestir de software a sus propios microordenadores. Algunas

La estructura en red exige la presencia de un nodo maestro, con unos requerimientos mínimos que permitan un eficaz control de las comunicaciones con los restantes nodos del sistema.



La "edad" de los ficheros

El MP/M es, precisamente, uno de los muchos sistemas operativos que poseen esta habilidad. Al efecto, cuenta con los comandos "SET" y "SDIR".
La actualización de la fecha de un archivo puede llevarse a cabo según varios criterios. Por ejemplo, en el caso del MP/M, es posible asociar la fecha actual a un archivo tanto en el instante de su creación, como al modificarlo, o incluso en cualquier momento que se acceda a un registro de dicho archivo. El comando SET es la herramienta que servirá para optar por uno u otro criterio, o por varios a la vez. Así pues, se pueden ejecutar los tres comandos siguientes:

A > *SET [CREATE = ON]*: añade fecha a los archivos creados.
A > *SET [UPDATE = ON]*: añade la fecha a los archivos modificados.
A > *SET [ACCESS = ON]*: añade fecha a los archivos accedidos.

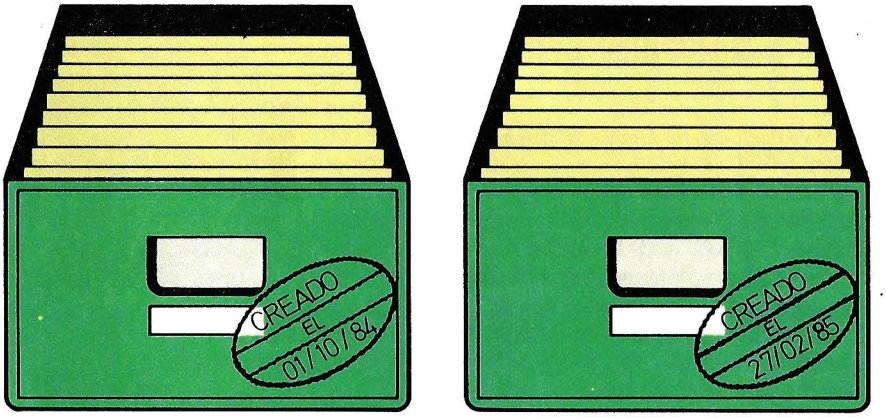
Una vez que se han definido las circunstancias en las que se ha de añadir o modificar la fecha de un fichero, puede ya entrar en escena el comando destinado a visualizar la situación de un fichero: SDIR. Suponga que desea conocer la fecha de creación y modificación de dos ficheros cuyos nombre son DATOS.DAT y RESULTAD. DAT, localizados en el directorio del disco B. Los pasos necesarios serán los siguientes:
— Indicar al sistema los modos de representación de la fecha de los ficheros.

A > *SET [CREATE = ON]*
A > *SET [UPDATE = ON]*

— Listar el directorio del disco en el que se encuentren los archivos en cuestión:

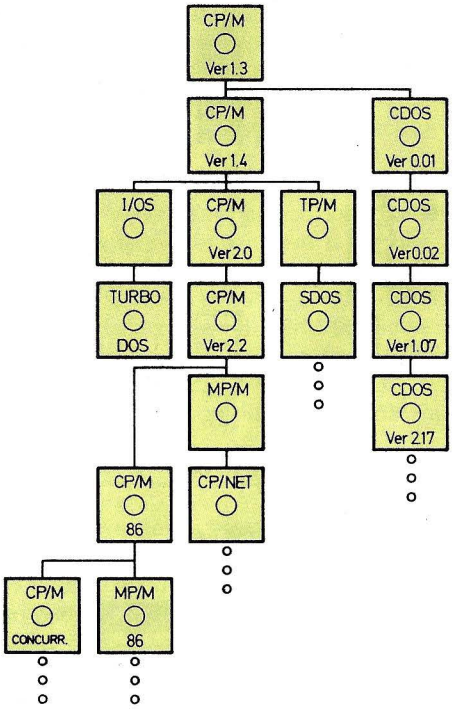
A > *SDIR B*

La respuesta del ordenador será:



Nombre	Bytes	Regs	Atributos	Prot	Modificación	Creación
RESULTAD.DAT	76K	608	Dir RW	None	01/01/85 10: 00	01/01/85 10: 00
DATOS.DAT	12K	95	Dir RW	None	01/01/85 08: 00	08/10/84 12: 00

En general, la respuesta que proporciona el sistema operativo incluye otros datos relativos a los ficheros: su tamaño en bytes, el número de registros que ocupan, el método de acceso permitido, las protecciones existentes, ...
Así pues, no cabe duda que es posible evaluar la edad de un fichero —distancia entre su fecha de creación y la actual— y, en consecuencia, conocer cuáles son los archivos que están en su "infancia", "juventud" o "vejez".
Cuando un sistema operativo es capaz de disponer de la fecha y la hora en curso, tiene en su mano una base de referencia de tiempos que le permitirá desarrollar una faceta insólita: "ser su propio autobiógrafo".
Aplicando esta facultad al conjunto de los ficheros del sistema, se puede llegar a tener una cronología histórica de los mismos, cronología que ayudará a su identificación y conservación. Un ejemplo indicativo puede ser el siguiente.
Durante una sesión de trabajo, el usuario se dedicó a modificar varios ficheros que contenían los programas en BASIC de diversas aplicaciones; si bien, al concluir la sesión, no llegó a finalizar la tarea por completo. Transcurridos varios días, el usuario desea reanudar la modificación de los programas; sin embargo, tropieza con el inconveniente de que no recuerda con exactitud cuáles fueron los ficheros modificados. El problema quedaría resuelto si dispusiera de un registro histórico de los ficheros que reflejara la fecha de modificación.



Arbol de evolución de la familia de sistemas operativos CP/M.

firmas han gozado de cierto éxito, entre ellas cabe destacar a Cromemco, Radio Shack, Zenith... A continuación se describen someramente algunos de estos sistemas operativos catalogables como derivados del CP/M.

CP/NET

Este nuevo sistema operativo está desarrollado para acceder al campo de los procesos distribuidos y a la comunicación remota entre diversos microordenadores integrados en una red.

Cabe considerarlo como un complemento en expansión del MP/M. En el caso del MP/M existía un único microordenador y múltiples usuarios, mientras que para el CP/NET existen múltiples microordenadores y múltiples usuarios, conectados entre sí, con la posibilidad de compartir cada uno de ellos los diferentes recursos del sistema. A diferencia con el MP/M, ningún usuario en particular asume el papel de sistema relegando a los demás a un plano de subordinación.

Con respecto a esta última precisión, cabe señalar que no es total la igualdad entre los componentes de una red. En efecto, debe existir algún elemento de la red que actúe como nodo maestro, responsable de coordinar a los restantes nodos de la red. Esta condición implica que el nodo maestro debe ser capaz de satisfacer unas exigencias mínimas, como son el que disponga de un sistema operativo MP/M y tenga libres para su uso al menos 16 Kbytes de memoria primaria.

Las ventajas principales de una red con respecto a un sistema multiusuario, pueden concretarse en la mayor velocidad de ejecución de los programas, al no tener que competir por una única CPU, y en una compartición total de los recursos de la red, ya sean propios del usuario o no.

CDOS

El sistema operativo CDOS fue desarrollado por Cromemco para equipar a los microordenadores de la propia firma. Su inspiración se encuentra en la versión más clásica del CP/M. Con respecto a éste presenta algunas diferencias derivadas de la total orientación del CDOS al microprocesador Z-80. Hay que tener en cuenta que el Z-80 posee un repertorio de códigos de operación bastante más dilatado que el microprocesador Intel 8080, protagonista de la versión más clásica del CP/M. La total compatibilidad entre ambos sistemas operativos tampoco es posible debido a que el CDOS incluye extensiones del sistema operativo propias; si un programa hace uso de ellas, es obvio que su ejecución no será posible bajo el control de otro sistema operativo distinto del CDOS.

I/OS

También conocido como TSA/OS, tiene su principal característica en la total compatibilidad con el CP/M-80 y el CDOS, mejorando incluso algunas de las propiedades de estos dos sistemas operativos.

TURBODOS

El objetivo primordial de este nuevo sistema operativo es, como revela su denominación, posibilitar un acceso a disco lo más rápido posible. Para ello, trata de calcular previamente las secciones del disco a las que va a acceder el usuario, reduciendo así los tiempos de lectura. Otra ventaja reseñable es que no exige la carga previa del sistema operativo para lograr un acceso a disco.

Las redes de microordenadores abren la posibilidad de compartir los recursos accesorios más caros del sistema: impresoras rápidas, digitalizadores, discos rígidos...

Visicalc (y 3)

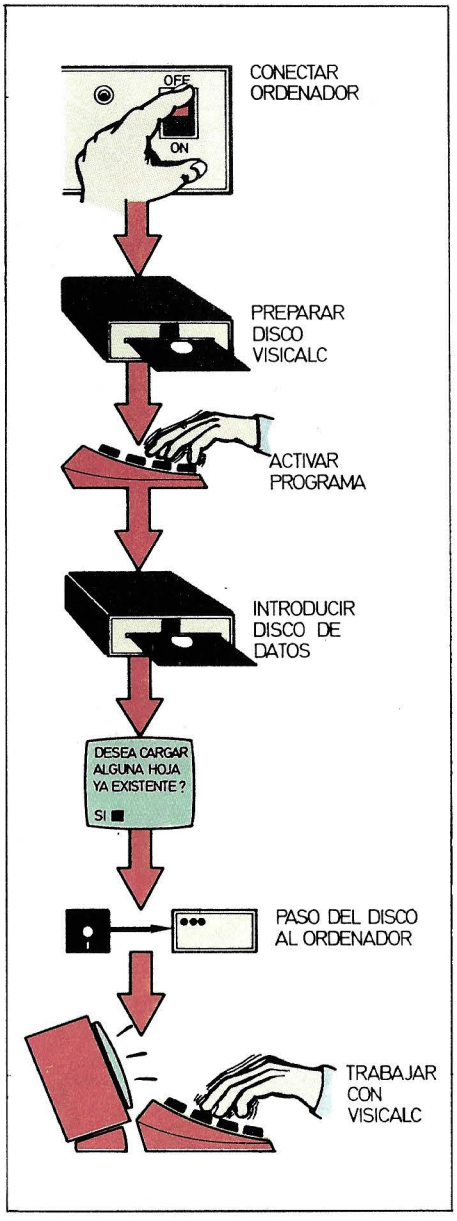
Una sesión de trabajo con la hoja electrónica

Para concluir el estudio con la hoja electrónica VISICALC, nada resulta más oportuno que acometer una sesión práctica; en ella, entrarán en juego los comandos y funciones más importantes. Mediante la resolución de algunos sencillos ejemplos, se vislumbrará claramente la mecánica de utilización del VISICALC. Desde luego, quedan fuera de esta sesión algunas propiedades importantes, que sólo podrían analizarse con detalle teniendo enfrente un ordenador personal y apoyándose en los manuales de aplicación.

PUESTA EN FUNCIONAMIENTO

Para comenzar una sesión de trabajo con el VISICALC, resulta imperativo introducir en la unidad correspondiente el disco que contiene al propio programa VISICALC. Una vez realizada esta operación hay que invocar al programa, con lo que de forma automática se entrará en el entorno de la hoja electrónica. En este instante deja de ser necesaria la utilización del disco flexible que soporta al programa y, con objeto de evitar posibles accidentes, es recomendable reemplazarlo por otro, previamente formateado, en el que se almacenarán (o del que se recuperarán) las matrices de cálculo.

Al terminar con las operaciones señaladas, aparecerá en la pantalla una matriz o retícula, constitutiva de la hoja electrónica. Las líneas estarán numeradas desde el 1 hasta el 21 y las columnas identificadas con las correspondientes letras a partir de la A. Ello significa que la ventana (pantalla de visualización) refleja la parte superior izquierda de la hoja electrónica. Si en ese momento se desea cargar alguna hoja ya creada, almacenada en disco tras concluir una sesión anterior se utilizará el comando "/SL". En caso contrario —si desea empezar con una hoja nueva—, puede ya iniciar sin mayor demora la sesión de trabajo.



Secuencia de operaciones necesarias para la puesta a punto del ordenador, dispuesto a trabajar con la aplicación de hoja electrónica VISICALC.

EJEMPLO 1: GASTO ORIGINADO POR DOS VEHICULOS

En el primer ejercicio práctico, se utilizará la hoja electrónica para controlar los gastos producidos por dos coches que se identificarán por medio de las letras A y B. Para ello, se tendrán en cuenta, por un lado, los kilómetros recorridos por cada coche, y por otro, los gastos totales producidos por cada uno de ellos. Estos últimos se desglosan en los apartados de combustible, reparaciones y mantenimiento. El objetivo es obtener el gasto que origina cada coche por kilómetro recorrido, y el gasto conjunto de ambos vehículos.

Normalmente, para definir el aspecto general que debe tener la hoja, se empieza utilizando el comando "/TH". En efecto, su colaboración permite definir los títulos horizontales; en nuestro caso, podemos inicializar el elemento A1 con el título "COCHE", el elemento B1 con "KM", el C1 con "COMBUSTIBLE", el D1 con "REPARACIONES", el E1 con "MANTENIMIENTO" y, por último, el F1 con "GASTOS/KM". A continuación, y también haciendo uso del comando "/TH", colocaremos todas las columnas de la segunda y quinta fila con títulos en blanco, para separar la cabecera de los datos. El último título a introducir, coincidirá con el elemento A6; en él teclearemos la palabra "TOTAL", con la misión de inicializar la línea de totales.

A continuación, pasamos a definir los elementos que hay que calcular; por supuesto, en función de otros elementos de la matriz. En nuestro caso, ello habrá que realizarlo de la siguiente forma:

1. Elemento F3=@SUM (C3, D3, E3)/B3
Con ello, se calcula la suma total de los gastos por combustible, reparaciones y mantenimiento del primer coche y se divide entre el número de kilómetros recorridos. De esta forma, se obtiene el gasto total por kilómetro del primer coche.

7. Elemento F6= @ SUM (C6, D6, E6)/B6
Evalúa el gasto total por kilómetro de los dos vehículos: se suman los totales de combustible, reparaciones y mantenimiento, y se divide dicha suma por los kilómetros recorridos.
Una vez realizada la asignación de funciones, cabe afirmar que disponemos ya del "esqueleto" de la hoja electrónica. Esta queda dispuesta para recibir a los datos en los elementos de entrada de la matriz. En el supuesto reflejado en la figura, se considera que el coche A es un SEAT 127, con 6.255 kilómetros recorridos y en el coche B es un SEAT IBIZA que ha recorrido 525 kilómetros, con un gasto de 15.613 pesetas en combustible, 17. 832 en reparaciones y 1.555 en mantenimiento. Sin más que introducir estos datos en los elementos de entrada, se producirá el cálculo automático de la columna F (gastos por kilómetro) y de la fila 6 (totales de ambos coches).
Una vez realizadas las anteriores operaciones, el usuario estará en disposición de utilizar el comando "/SQ", si desea terminar la sesión sin almacenar la hoja, o el comando "/SS" si quiere guardar la hoja para utilizarla en una nueva ocasión. Por supuesto, la hoja confeccionada será recalculada automáticamente cada vez que se modifique cualquiera de los datos aportados.

	A	B	C	D	E	F
1	COCHE	KM	COMBUS.	REPAR.	MANTENIM.	GASTOS/KM
2	ƀ	ƀ	ƀ	ƀ	ƀ	ƀ
3						@ SUM (C_3,D_3,E_3)/B_3
4						@ SUM (C_4,D_4,E_4)/B_4
5	ƀ	ƀ	ƀ	ƀ	ƀ	ƀ
6	TOTAL	@ SUM (B_3,B_4)	@ SUM (C_3,C_4)	@ SUM (D_3,D_4)	@ SUM (E_3,E_4)	@ SUM (C_6,D_6,E_6)/B_6

Estructura de la hoja electrónica creada para resolver el ejemplo 1: "gasto originado por dos automóviles".

	A	B	C	D	E	F
1	COCHE	KM	COMBUS.	REPAR.	MANTENIM.	GASTOS/KM
2						
3	SEAT 127	6.255	90.135	55.127	40.233	30
4	SEAT IBIZA	525	15.613	17. 832	1.555	67
5						
6	TOTAL	6.780	105.748	72.959	41.788	33

Aspecto de la hoja electrónica del ejemplo 1, una vez introducidos los datos necesarios.

2. Elemento F4= @ SUM (C4, D4, E4)/B4
De nuevo, se repite el cálculo realizado para el elemento precedente (F3); si bien, esta vez con objeto de obtener el gasto por kilómetro del segundo coche.

3. Elemento B6= @ SUM (B3, B4)
La simple suma de las células B3 y B4 obtiene los kilómetros recorridos entre los dos coches (ver figura adjunta).

4. Elemento C6= @ SUM (C3, C4)
Su finalidad es obtener en el elemento C6 la suma de los gastos de combustible de ambos coches.

5. Elemento D6= @ SUM (D3, D4)
Para determinar el gasto total por reparaciones de los dos vehículos.

6. Elemento E6= @ SUM (E3, E4)
De esta forma, se obtiene el importe total del mantenimiento de ambos coches.

que se han invertido 90.135 pesetas en combustible, 55.127 en reparaciones y 40.233 en mantenimiento. A su vez, el

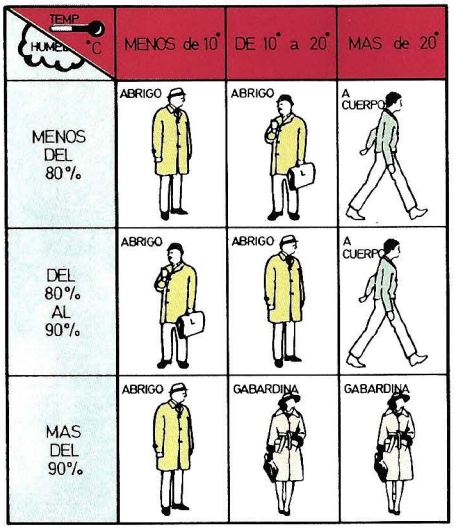


Tabla de doble entrada en la que se puede decidir la indumentaria apropiada, en función de la temperatura y humedad relativa.

EJEMPLO 2: DECISION AUTOMATICA

Para lograr una mayor familiaridad con las funciones lógicas —probablemente sean estas funciones las menos conocidas—, se describe a continuación un segundo caso práctico, en el que las funciones de este tipo juegan un papel importante. Suponga que pretendemos que la hoja electrónica nos ayude a decidir si para salir a la calle es conveniente coger el abrigo, la gabardina o ir a "cuerpo libre". Una decisión que se tomará de acuerdo a la temperatura y humedad relativa del aire.

- En el caso de que la temperatura sea

	A	B
1	TEMPERATURA	
2	HUMEDAD RELATIVA	
3		
4	ABRIGO	@ OR (B_1<10, @ AND (B_1<21, B_2>90))
5	GABARDINA	@ AND (B_1>9, B_2>89)
6	A CUERPO	@ AND (B_1>20, B_2<90)

Aspecto del "esqueleto" de la hoja electrónica creada para la resolución del ejemplo 2.

inferior a 10° C, sin duda saldremos con abrigo.

- Si la temperatura está comprendida entre 10° y 20° C, tomaremos el abrigo cuando la humedad relativa del aire sea inferior al 90 %; en cambio, si es superior o igual al 90 % cogeremos la gabardina.
- Por último, si la temperatura es superior a 20° C, no utilizaremos prenda alguna cuando la humedad relativa sea menor del 90 % y tomaremos la gabardina en caso contrario.

El problema cabe plantearlo, desde un punto de vista lógico, en base a dos variables numéricas de entrada: temperatura y

	A	B
1	TEMPERATURA	15
2	HUMEDAD RELATIVA	95
3		
4	ABRIGO	FALSO
5	GABARDINA	CIERTO
6	A CUERPO	FALSO

Presentación en pantalla de la hoja electrónica del ejemplo 2 tras introducir la información apropiada.

Teoría de funciones (y 2)

En las funciones de un único valor y una única variable, la variable incógnita que permite deducir el resultado final de la función puede ser, a su vez, otra función. Ciertamente, éste es un hecho que se verifica en el caso general; no obstante, para simplificar el ejemplo práctico que se acompaña, la demostración se reduce a la situación enunciada. Esta circunstancia obedece, en el campo matemático, al apelativo de "composición de funciones". Desde luego, la composición de funciones puede ser perfectamente utilizada en una hoja electrónica.

Suponga que los elementos 1-A y 1-B de una hoja electrónica almacenan los ingresos anuales de los dos componentes de un matrimonio. A su vez, considere que el elemento 1-C está definido, en función de los anteriores, de la siguiente forma:

ELEMENTO 1-C=ELEMENTO 1-A+ ELEMENTO 1-B

En términos más rigurosos, cabe denominar "f" a la función; con ello, el elemento 1-C quedará definido como sigue:

ELEMENTO 1-C=f (ELEMENTO 1-A, ELEMENTO 1-B)

La expresión indica que el valor del elemento 1-C variará, según la función f, cuando varíe cualquiera de los elementos 1-A ó 1-B. Esto es: los ingresos de la familia varían cuando lo hace el ingreso procedente de alguno de sus dos componentes.

Suponga ahora que en el elemento 1-D se desea obtener el 18 % del ingreso total del matrimonio; por ejemplo, para previsión de impuestos. En tal caso, puede definirse el elemento 1-D en función del 1-C; por ejemplo:

$$\text{ELEMENTO 1-D} = \frac{18 \times \text{ELEMENTO 1-C}}{100}$$

Si a esta nueva función la denominamos g, la definición del referido elemento será:

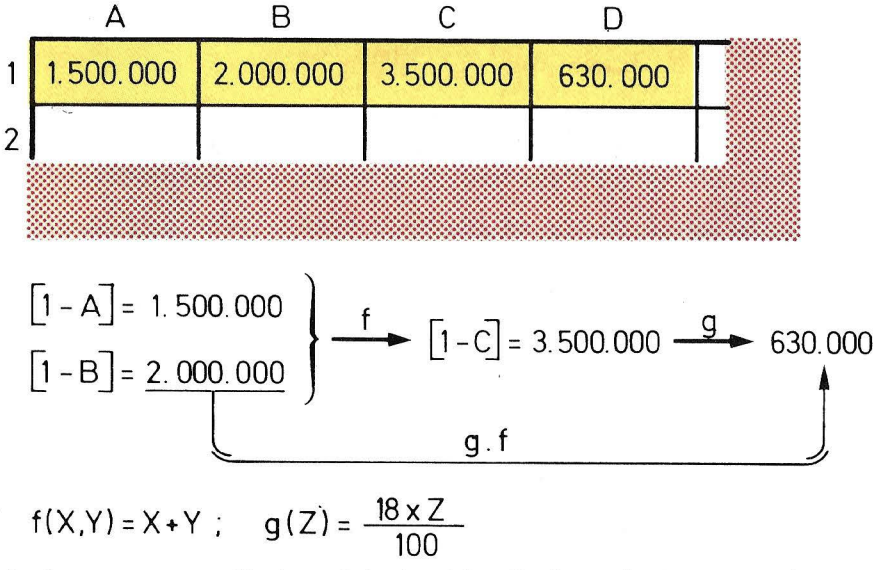
ELEMENTO 1-D=g (ELEMENTO 1-C)

Ello indica que el valor del elemento 1-D variará según la función g, cuando varíe el elemento 1-C. En definitiva: el porcentaje previsto para impuestos varía cuando lo hace el ingreso total de la familia.

Analizando las funciones anteriores, se llega a la conclusión de que el cálculo del importe previsto para impuestos puede realizarse mediante la siguiente función compuesta:

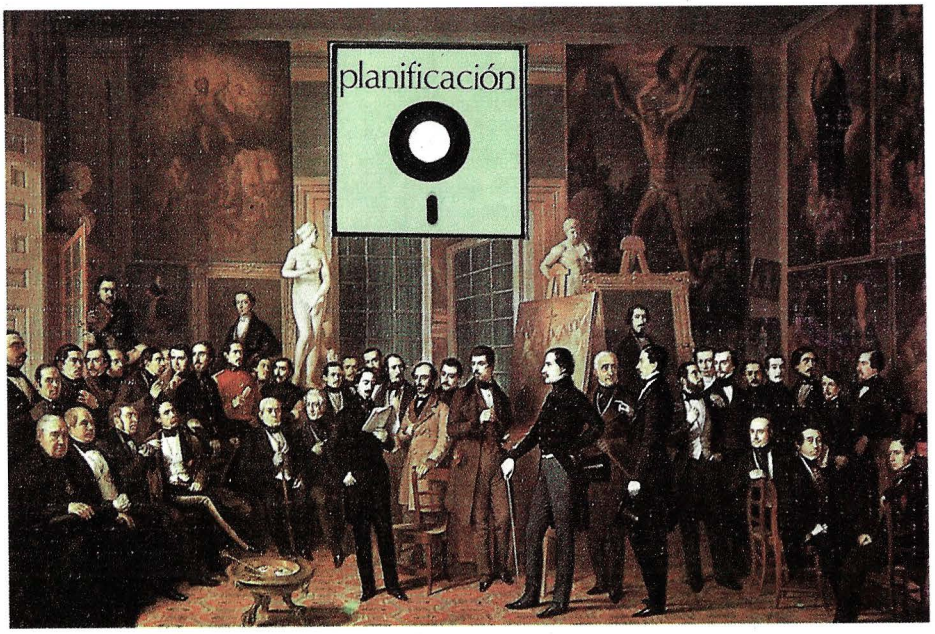
ELEMENTO 1-D=g (f (ELEMENTO 1-A, ELEMENTO 1-B))

De esta forma, se representa que aplicando la función f a los elementos 1-A y 1-B, y aplicando la función g al resultado anterior, se obtiene el resultado definitivo. Al margen de su razonamiento teórico, la composición de funciones es una técnica utilizable para confeccionar expresiones de cálculo destinadas a los elementos de una hoja electrónica. El gráfico adjunto ilustra el ejemplo analizado.

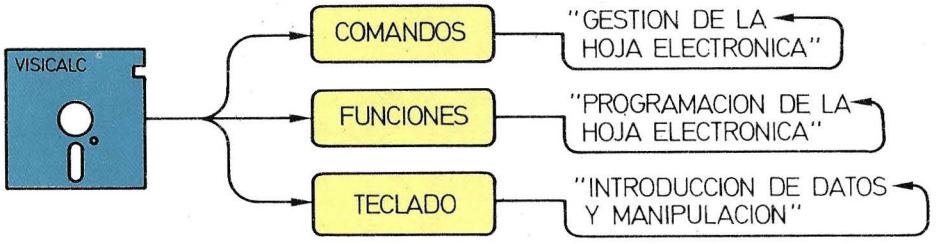


$$f(X,Y) = X + Y \; ; \quad g(Z) = \frac{18 \times Z}{100}$$

La figura muestra una fila de una hoja electrónica. En ella, su elemento 1-D puede calcularse mediante una función sencilla (g), dependiente del elemento 1-C, o mediante una función compuesta (g f), dependiente de los elementos 1-A y 1-B.



La planificación y el análisis de supuestos del tipo "What if..?" —¿Qué ocurriría si..?— son tareas en las que las hojas electrónicas se revelan como herramientas de indudable eficacia.



Estructura básica del paquete de aplicación VISICALC.

humedad relativa del aire; y a tres variables lógicas de salida: "abrigo", "gabardina" y "a cuerpo". De estas últimas, dos adoptarán el valor FALSO y la tercera VERDADERO. Las condiciones para la toma de decisión, según han sido expuestas anteriormente, aparecen resumidas en el cuadro adjunto.
Para resolver el problema, sólo queda por decidir qué elementos de la matriz se utilizarán en cada supuesto. Una posible solución es la que se relaciona a continuación:

1. Temperatura:
— literal en la posición A1
— valor en la posición B1

2. Humedad relativa:
— literal en la posición A2
— valor en la posición B2

3. Abrigo:
— literal en la posición A4
— decisión en la posición B4

4. Gabardina
— literal en la posición A5
— decisión en la posición B5

5. A "cuerpo":
— literal en la posición A6
— decisión en la posición B6

Si la aplicación VISICALC se encuentra ya cargada en la memoria del ordenador, pueden dar comienzo las operaciones necesarias para definir la hoja electrónica adecuada en este caso práctico. En primer lugar, se empleará como comando "/TH" para introducir los literales: "TEMPERATURA", "HUMEDAD RELATIVA", "ABRIGO", "GABARDINA" y "A CUERPO" en las posiciones A1, A2, A4, A5 y A6, respectivamente.
En la fila 3 se introducirá una línea, a base de guiones, para separar la zona de entrada de datos respecto a la zona de salida para las decisiones lógicas.
Una vez introducidos los títulos, pueden ya definirse las funciones lógicas de los elementos B4, B5 y B6 (ver figura):

1. Elemento B4= @ OR (B1<10, @ AND (B1<21,B2<90))
La expresión lógica especifica que B4 sólo será VERDADERO si B1<10 (la temperatura es inferior a 10° C), o cuando la temperatura sea menor o igual a 20° C (B1<21) y, además, la humedad relativa sea menor del 90 % (B2<90).

2. Elemento B5= @ AND (B1>9,B2>89)
En este caso, B5 será VERDADERO "si y sólo si" la temperatura es mayor o igual a 9° C (B1>9) y además la humedad relativa es superior al 89 % (B2>89).

3. Elemento B6= @ AND (B1>20, B2<90)
B6 sólo tomará el valor VERDADERO cuando la temperatura supere los 20° C (B1>20) y la humedad relativa sea inferior al 90 % (B2<90).

En estas condiciones, si se introduce el valor 15 en el elemento B1 y el dato 95 en el elemento B2, la hoja electrónica aconsejará, prudentemente, que utilicemos la gabardina. Exactamente, su respuesta será: "ABRIGO": FALSO, "GABARDINA": VERDADERO, "A CUERPO": FALSO.

RECAPITULACION SOBRE EL VISICALC

A raíz del estudio de la aplicación VISICALC, acometido en los tres últimos capítulos de la obra, se llega a la conclusión de que este paquete participa de todas las características típicas de una hoja electrónica. Su lenguaje de manipulación, tanto a nivel de comandos como de funciones, permite un empleo sencillo y eficiente, al alcance de cualquier usuario no especializado.
El usuario especializado tal vez no aplauda la escasa complejidad y amplitud del lenguaje de funciones. En cualquier caso, ésta es una característica compartida por todas las aplicaciones de la hoja electrónica. El motivo hay que buscarlo en que estos productos no están diseñados para realizar una programación clásica, sino para resolver problemas cotidianos que no exigen una programación compleja.

Datos alfanuméricos

Tratamiento de cadenas de caracteres

Un ordenador es algo más que una máquina que permite operar con números a gran velocidad. El ordenador también es capaz de procesar otro tipo de información, por ejemplo datos alfanuméricos. Los datos alfanuméricos son secuencias de caracteres (letras, números y símbolos especiales) colocados uno tras otro. En la memoria del ordenador estos caracteres se almacenan mediante sus correspondientes códigos ASCII.

Los caracteres que forman una cadena alfanumérica pueden ser letras mayúsculas o minúsculas, cifras, signos de puntuación, y símbolos algebraicos o de relación. Los datos de tipo alfanumérico pueden manejarse a modo de constantes o por medio de variables. Una constante alfanumérica ha de especificarse encerrando los caracteres que la componen entre comillas. Los siguientes son ejemplos válidos de constantes alfanuméricas:

```
"PEPE"
"U277"
"CARACOL"
"A+2="
```

En realidad, este tipo de datos no es nuevo, ya han sido empleados para visualizar mensajes en la pantalla a través del comando PRINT:

```
PRINT "PRIMER CASO"
```

La instrucción precedente comunica al ordenador que imprima una constante alfanumérica constituida por la siguiente secuencia de caracteres:

```
P,R,I,M,E,R,<espacio>,C,A,S,O
```

Aunque tampoco es ésta la única operación que puede efectuar el ordenador con datos alfanuméricos. También puede manejar variables de este tipo.

Una variable alfanumérica tiene el mismo cometido que una variable numérica. En general, una variable es una zona de almacenamiento. En ella se guarda un valor susceptible de ser modificado durante la ejecución de un programa. Al emplear una variable, el ordenador reserva un espacio para su almacenamiento. Tras ello, es posible utilizar dicha zona para almacenar los sucesivos valores asignados a la referida variable.

Para distinguir una variable alfanumérica

El lenguaje BASIC es capaz de procesar dos tipos de datos: numéricos y cadenas de caracteres. Ambos reciben un tratamiento distinto por parte del ordenador.

de otra que no lo es, es necesario terminar su nombre con el símbolo "dólar" ($). A continuación se exponen algunos ejemplos de variables de esta naturaleza:

```
A$="ALUMNO"
B$="ES"
C$="JUAN"
D$="UN"
```

Ello nos permitirá realizar algunas combinaciones con la instrucción PRINT. Por ejemplo:

```
PRINT C$;B$;D$;A$
```

Cuyo resultado en la pantalla es el siguiente:

```
JUAN ES UN ALUMNO
```

Es preciso diferenciar los datos alfanuméricos de los numéricos, puesto que son de distinta naturaleza y son tratados por la máquina de distinta forma. Así, por ejemplo, una asignación como la siguiente:

```
B$=234
```

sería rechazada con la presencia de un mensaje de error del tipo "TYPE MISMATCH" o similar. Con ello, el ordenador indica que la asignación es incorrecta, ya que se intenta mezclar dos tipos distintos de datos. Tal incorrección puede arreglarse poniendo comillas en el lugar adecuado:

```
B$="234"
```

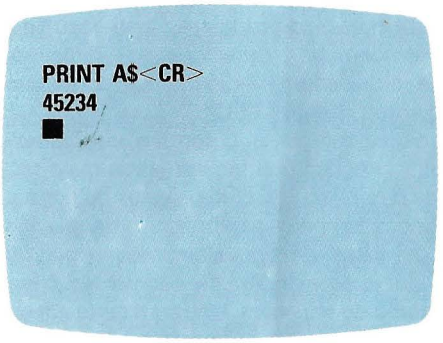
La nueva asignación será aceptada por el ordenador sin ningún problema. No obstante, hay que tener en cuenta que la variable B$ no tiene asignado el número 234, sino la secuencia de caracteres 2, 3, 4: lo que significa que no se pueden realizar operaciones algebraicas con dicha variable. Así pues, la siguiente instrucción:

```
A=45+B$
```

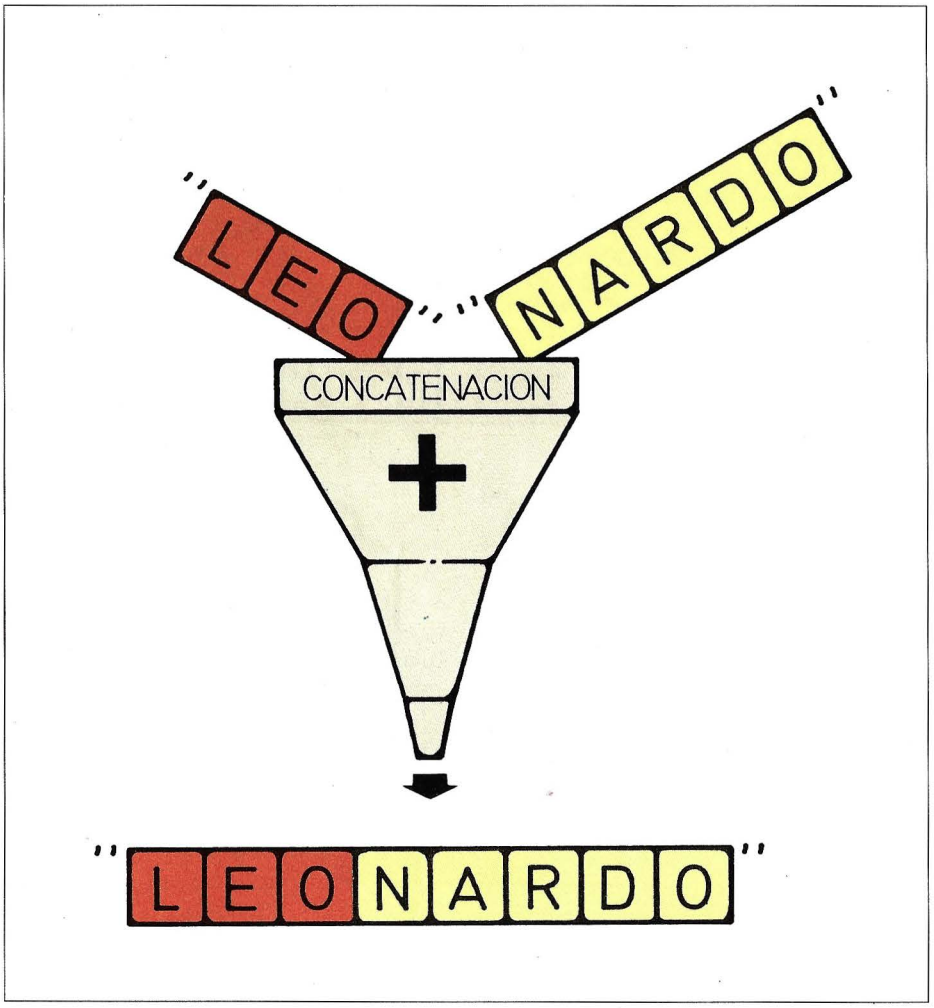
será rechazada con el mismo mensaje de error que el indicado en un párrafo anterior. En efecto, no es posible realizar la suma de una variable numérica con otra no numérica. Ante tal perspectiva, se podría probar la siguiente solución:

```
A$="45"+B$
```

En principio, no aparece ningún mensaje de error, lo que quiere decir que la operación es correcta. La sorpresa surge al visualizar el contenido actual de la variable A$.



La respuesta no coincide con la suma de los números 45 y 234. ¿Qué ha sucedido? Sencillamente que la suma de dos cadenas de caracteres no es equivalente a una operación aritmética, sino a la "concatenación" de ambas; esto es: a la unión de las dos cadenas, una a continuación de la otra.
Desde luego, esta operación no es conmutativa, ya que no es lo mismo PACO-PEPE que PEPEPACO.



La concatenación de cadenas de caracteres es la operación básica realizable sobre datos de tipo alfanumérico. Consiste, sencillamente, en la unión de las cadenas componentes.

FRACCIONAMIENTO DE CADENAS

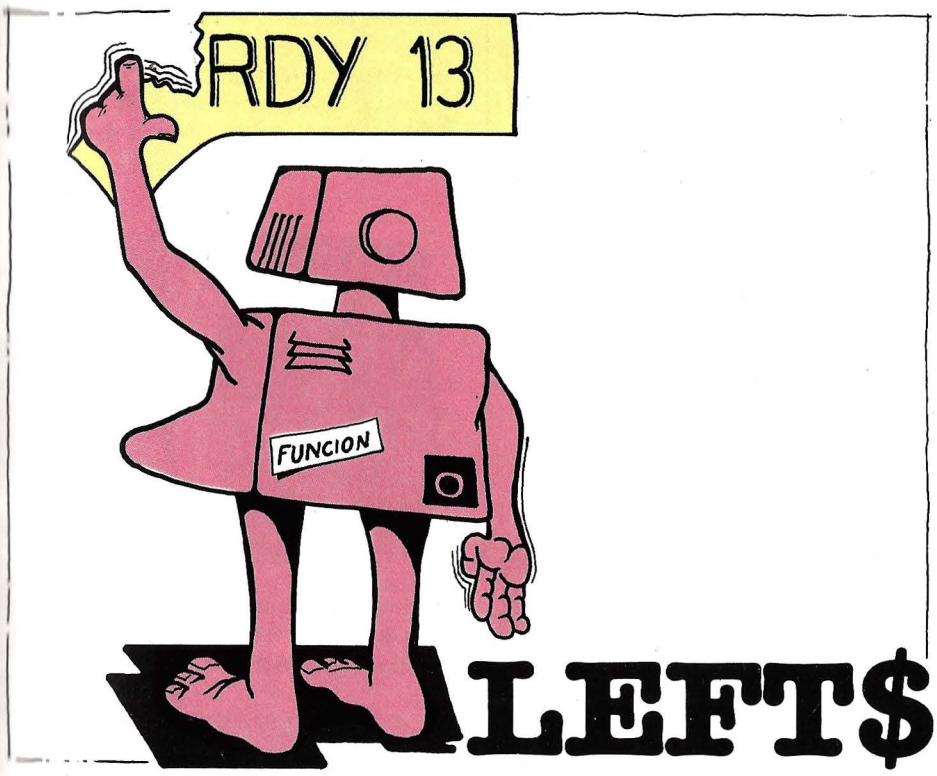
La posibilidad de unir dos o más cadenas para formar una nueva (concatenación de cadenas), permite construir distintas combinaciones partiendo de varias cadenas componentes. El siguiente caso, opuesto

LEFT$

Extrae los N caracteres situados más a la izquierda de la cadena sobre la que actúa. Se trata de una función, de ahí que deba actuar en combinación con un comando.

Formato: *LEFT$ (<argumento>, N)*
<argumento>: cadena de caracteres afectada.
N: número natural.

Ejemplos: PRINT LEFT$ ("CASA", 2)
LET B$=LEFT$ (A$, 5)



al anterior, radica en la posibilidad de extraer fragmentos de una cadena; esto es, separar una parte o subconjunto de la misma. Para ello el BASIC dispone de tres funciones: LEFT$, RIGHT$ y MID$.
La primera de ellas resulta adecuada para extraer los elementos situados a la izquierda de la cadena de caracteres (no hay que olvidar que la palabra LEFT significa izquierda en inglés).
El formato de esta función es el siguiente:

LEFT$ (<argumento>,N)

La zona <argumento> contiene la cadena sobre la que se realiza la operación. Esta puede expresarse como dato constante, o bien, puede incluir a una variable representativa de la cadena de caracteres afectada. A su vez, N es un número natural que especifica el número de caracteres a extraer de la cadena, por supuesto, a partir de su extremo izquierdo. Las siguientes instrucciones incluyen la función definida:

```
PRINT LEFT$("SOFTWARE",5)
A$=LEFT$("CASA",2)
C$=LEFT$(S$,3)
```

Hay que hacer notar que LEFT$ es una función y, por lo tanto, ha de utilizarse asociada a un comando. De lo contrario, el ordenador no sabría qué hacer con el resultado obtenido. Por ejemplo, la función LEFT$ puede utilizarse dentro de una instrucción PRINT:

```
PRINT LEFT$ ("JUANJOSE",4) <CR>
JUAN
■
```

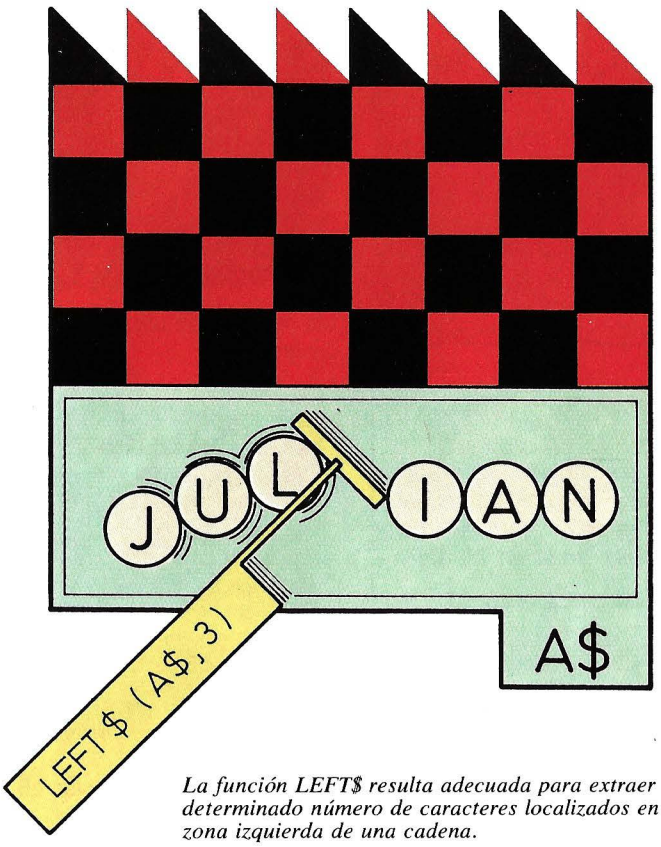
Un nuevo ejemplo lo aporta el siguiente programa:

```
10 B$="MARICARMEN"
20 C$=LEFT$ (B$,4)
30 PRINT C$
40 END

RUN <CR>
MARI
■
```

En ambos casos, la función LEFT$ se emplea para extraer los cuatro caracteres situados más a la izquierda de las cadenas indicadas. El número N con el que se define el número de caracteres a extraer de la cadena, puede ser sustituido también por una variable; en tal caso, se extraerá el número de caracteres que señale el contenido de la variable en cuestión.

```
10 B$="12345678"
20 FOR I=1 TO 8
30 PRINT LEFT$(B$,I)
40 NEXT I
50 END
■
```

Este programa utiliza la última posibilidad señalada para realizar un tratamiento más complejo de la cadena B$. Por medio de un bucle FOR/NEXT, se altera el valor de I (número de caracteres a extraer), de tal forma que, en sucesivos ciclos, se construyan subcadenas de uno, dos, tres, ... hasta ocho caracteres.



La función LEFT$ resulta adecuada para extraer un determinado número de caracteres localizados en la zona izquierda de una cadena.

```
RUN
1
12
123
1234
12345
123456
1234567
12345678
■
```

Desde luego, también es posible extraer un conjunto de caracteres situados a la derecha de una determinada cadena. Para realizar esta función, hay que recurrir a la función RIGHT$. Su formulación es la siguiente:

RIGHT$(<argumento>,N)

De nuevo, en el campo denominado argumento se especifica la cadena, en forma de constante o variable, mientras que N es el número de caracteres que se desea extraer. Por ejemplo:

```
PRINT RIGHT$("TITO",2)
B$=RIGHT(B$,5)
```

Ambas son instrucciones que hacen un uso correcto de la función RIGHT$. Al igual que LEFT$, RIGHT$ es también una función, por lo que las consideraciones que se hicieron anteriormente son extensivas a este caso. Por ejemplo, una instrucción como la que sigue:

```
RIGHT$("TIO",2)
```

obtendría como respuesta un mensaje de error: no hay que perder de vista que una función debe operar siempre asociada a un comando.

```
PRINT RIGHT$("TRES", 2) <CR>
ES
■
```

A continuación aparece un ejemplo en el que la función RIGHT$ se introduce dentro de un breve programa.

```
10 B$="FRANCISCO"
20 PRINT RIGHT$(B$,5)
30 END
RUN
CISCO
■
```

Su ejecución ilustra el efecto de RIGHT$(B$,5): extraer los cinco caracteres situados más a la derecha de la cadena de caracteres B$.
También es posible en este caso representar por medio de una variable el número de caracteres a recoger de la cadena. El siguiente programa ilustra un posible resultado.

RIGHT$

Extrae los N caracteres situados más a la derecha de la cadena.

Formato: *RIGHT$ (<argumento>, N)*
<argumento>: cadena de caracteres.
N: número natural.

Ejemplos: PRINT RIGHT$ (B$, 4)
B$=RIGHT$ ("CASUCHA", 3)

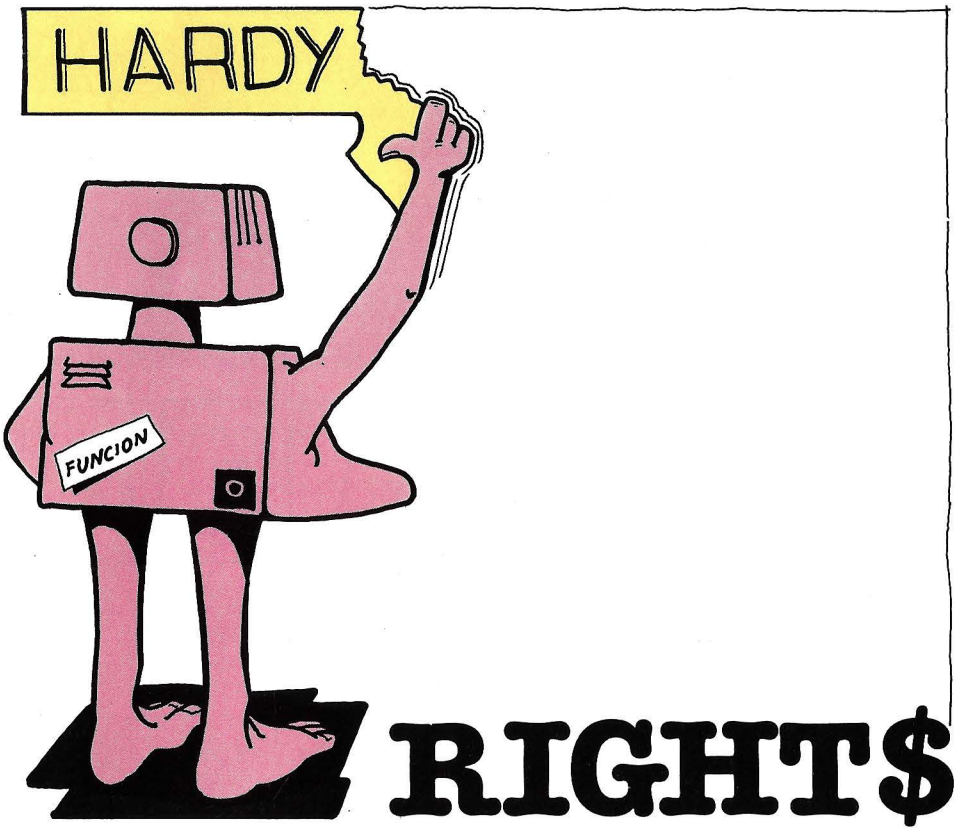
```
10 C$="123456789"
20 FOR I=1 TO 9
30 PRINT RIGHT$(C$,I)
40 NEXT I
50 END
■
```

Su ejecución creará sucesivas subcadenas, tomando de C$ los caracteres que indique el contenido de la variable I:

```
RUN
9
89
789
6789
56789
456789
3456789
23456789
123456789
```

Desde luego, ambas funciones permiten un amplio abanico de posibilidades, pero dejan abierta una cuestión: ¿es posible extraer un conjunto de caracteres que no estén situados a los extremos de la cadena? La respuesta es sí; para ello hay que emplear una nueva función denominada MID$. Su formato es el siguiente:

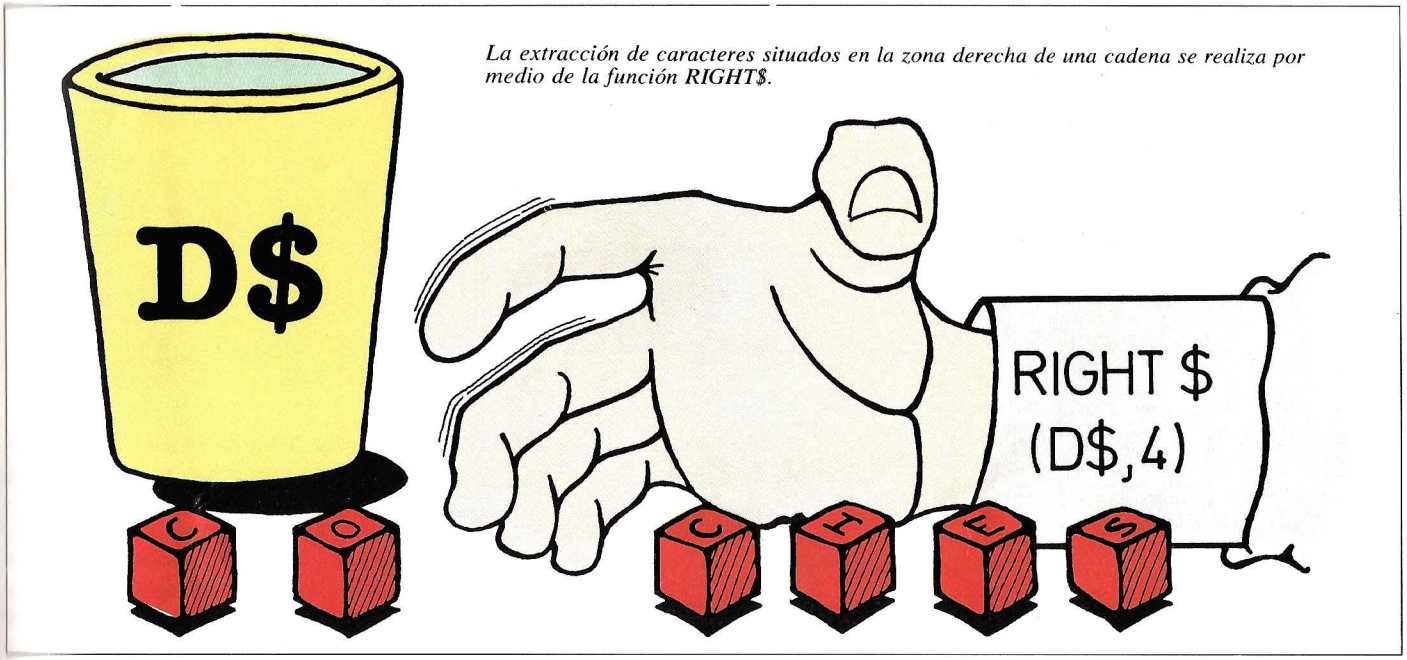
MID$(<argumento>,N,M)



RIGHT$

La cadena sobre la que ha de actuar se especifica en la zona <argumento>. N indica el carácter a partir del cual se desea efectuar la extracción, y M señala el número de caracteres a extraer.

La función opera colocando el punto de "corte" de la cadena en el carácter situado en la posición N, contada a partir de la izquierda, y extrae los M caracteres siguientes. Veamos un ejemplo práctico.



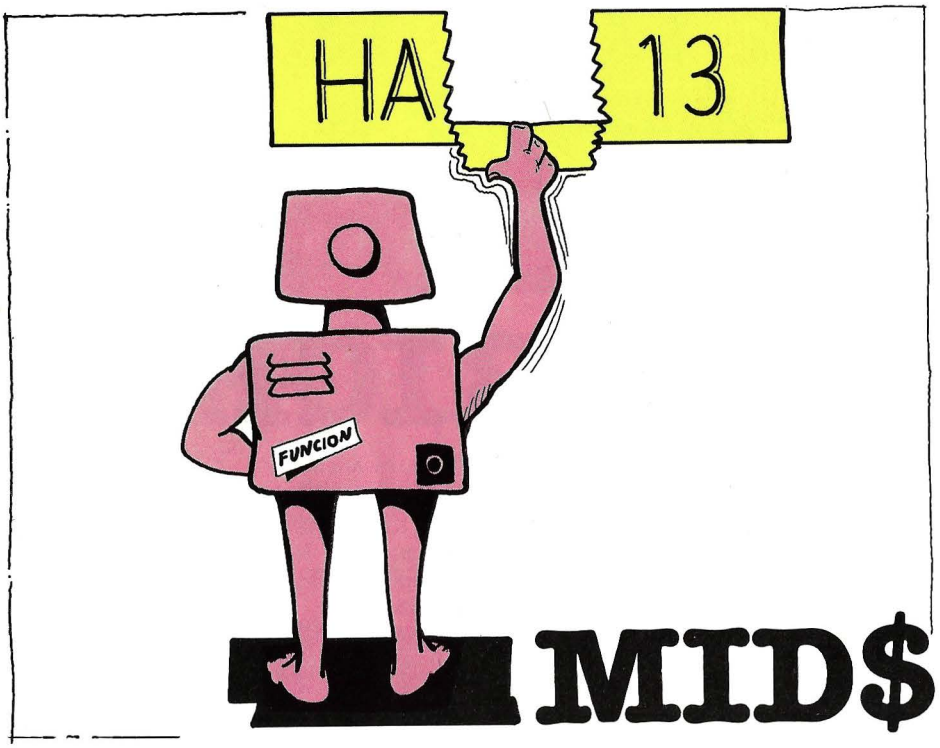
La extracción de caracteres situados en la zona derecha de una cadena se realiza por medio de la función RIGHT$.

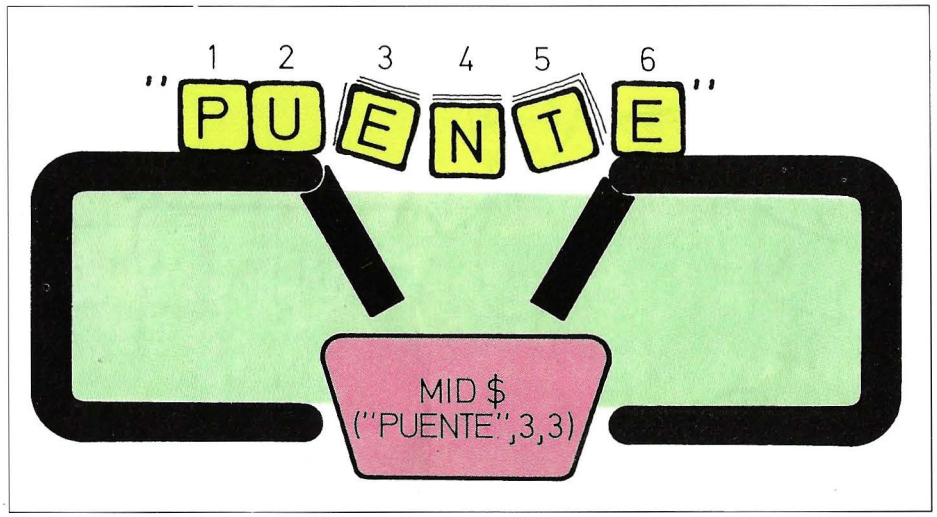
MID$

Extrae un grupo de M caracteres de la cadena indicada, tomados a partir del carácter que ocupa la posición N (inclusive).

Formato: *MID$ (<argumento>, N, M)*
<argumento>: cadena de caracteres sobre la que actúa.
N y M: números naturales.

Ejemplos: PRINT MID$ ("CADENA", 3, 2)
B$=MID$ (C$, 5, 5)





La función MID$ permite tomar caracteres localizados en las posiciones internas de una cadena.

```
PRINT MID$("CADENA",3,2) <CR>
DE
■
```

El punto de corte se sitúa a partir del tercer carácter, y se extraen los dos caracteres que siguen.
Esta función, empleada adecuadamente, permite extraer por separado todos y cada uno de los caracteres de una cadena alfanumérica.

```
10 C$="123456789"
20 FOR I=1 TO 9
30 PRINT MID$ (C$,I,1)
40 NEXT I
50 END
■
```

El programa en cuestión mostrará en la pantalla la siguiente secuencia:

```
RUN
1
2
3
4
5
6
7
8
9
■
```

Como se observa en el anterior ejemplo, MID$ admite también el uso de variables sustituyendo a los índices M y N.
Una de las aplicaciones del tratamiento de cadenas es la comprobación de mensajes destinados a mantener un diálogo hombre-máquina. Quizás, el caso más sencillo sea la simple comprobación

TABLA DE CONVERSION

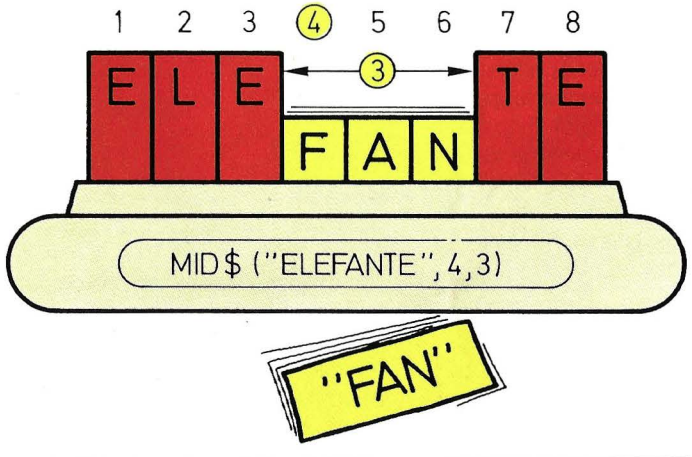
ORDENADOR	CONCATENACION	LEFT$	RIGHT$	MID$
	<c1>+<c2>	LEFT$ (<c>, <n>)	RIGHT$ (<c>, <n>)	MID$ (<c>, <p>, <n>)
APPLE II (APPLESOFT)	<c1>+<c2>	LEFT$ (<c>, <n>)	RIGHT$ (<c>, <n>)	MID$ (<c>, <p>, <n>)
APRICOT (M-BASIC)	<c1>+<c2>	LEFT$ (<c>, <n>)	RIGHT$ (<c>, <n>)	MID$ (<c>, <p>, <n>)
ATARI	—	—	—	<c> (<p>, <q>)
CBM 64	<c1>+<c2>	LEFT$ (<c>, <n>)	RIGHT$ (<c>, <n>)	MID$ (<c>, <p>, <n>)
DRAGON	<c1>+<c2>	LEFT$ (<c>, <n>)	RIGHT$ (<c>, <n>)	MID$ (<c>, <p>, <n>)
EQUIPOS MSX	<c1>+<c2>	LEFT$ (<c>, <n>)	RIGHT$ (<c>, <n>)	MID$ (<c>, <p>, <n>)
HP-150	<c1>+<c2>	LEFT$ (<c>, <n>)	RIGHT$ (<c>, <n>)	MID$ (<c>, <p>, <n>)
IBM PC	<c1>+<c2>	LEFT$ (<c>, <n>)	RIGHT$ (<c>, <n>)	MID$ (<c>, <p>, <n>)
MPF	<c1>+<c2>	LEFT$ (<c>, <n>)	RIGHT$ (<c>, <n>)	MID$ (<c>, <p>, <n>)
NCR DM-V (MS-BASIC)	<c1>+<c2>	LEFT$ (<c>, <n>)	RIGHT$ (<c>, <n>)	MID$ (<c>, <p>, <n>)
NEW BRAIN	<c1>+<c2>	LEFT$ (<c>, <n>)	RIGHT$ (<c>, <n>)	MID$ (<c>, <p>, <n>)
ORIC	<c1>+<c2>	LEFT$ (<c>, <n>)	RIGHT$ (<c>, <n>)	MID$ (<c>, <p>, <n>)
SHARP MZ-700 (MZ-BASIC)	<c1>+<c2>	LEFT$ (<c>, <n>)	RIGHT$ (<c>, <n>)	MID$ (<c>, <p>, <n>)
SINCLAIR QL	&	—	—	<c> (<p> TO <q>)
SPECTRAVIDEO	<c1>+<c2>	LEFT$ (<c>, <n>)	RIGHT$ (<c>, <n>)	MID$ (<c>, <p>, <n>)
ZX-SPECTRUM	<c1>+<c2>	—	—	<c> (<p> TO <q>)

<c1>, <c2>, <c>: Cadena alfanumérica con la que se ha de realizar la operación. <n>: Número de caracteres a extraer. <p>: Número de orden del primer carácter a extraer. <q>: Número de orden del último carácter a extraer.

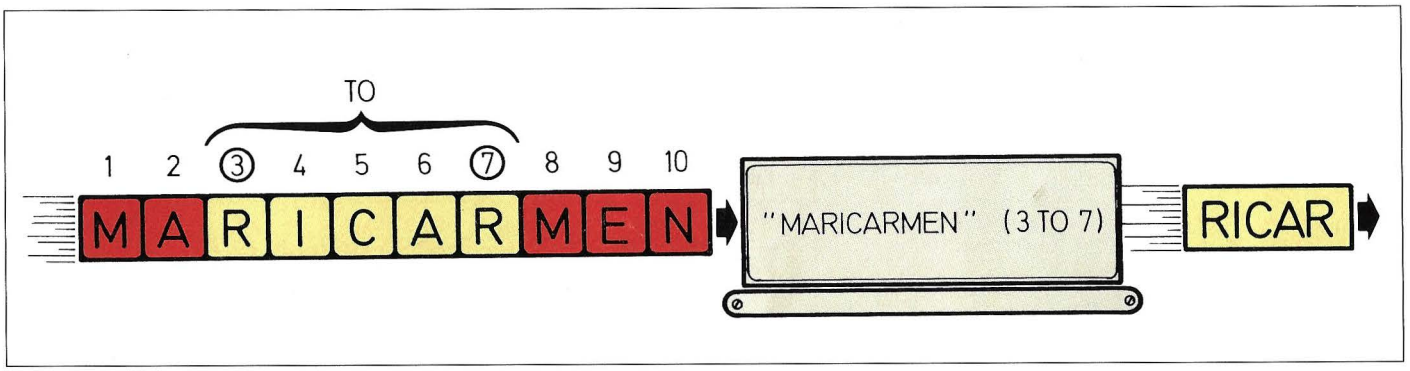
de si la respuesta dada a una pregunta es SI o NO. Veamos la primera posibilidad:

```
10 INPUT"DESEA VER EL RESULTADO";B$
20 IF B$="SI" THEN GOTO 100
30 IF B$="NO" THEN GOTO 200
40 PRINT "ERROR"
50 GOTO 10
100 PRINT "RESULTADO";2+2
200 END
```

Esta es una zona de decisión muy frecuente en los programas BASIC. A la pregunta señalada en la línea 10, hay que responder afirmativa o negativamente. Las instrucciones 20 y 30 comprueban si la respuesta es SI o NO y bifurcan la ejecución según corresponda.



Al utilizar la función MID$, hay que especificar dos valores: la posición que ocupa en la cadena el primer carácter afectado y el número de ellos que hay que extraer del dato en cuestión.



En los ordenadores de la firma Sinclair —tal es el caso del popular ZX-SPECTRUM—, las funciones habituales para la extracción de caracteres son reemplazadas por la función genérica TO. Los dos valores que la acompañan identifican al primer y al último caracteres a extraer.

Complicando un poco más el programa, éste puede llegar a admitir abreviaturas en la respuesta:

```
 10 INPUT"DESEA VER EL RESULTADO";B$
 20 C$=LEFT$(B$,1)
 30 IF C$="S" THEN GOTO 100
 40 IF C$="N" THEN GOTO 200
 50 PRINT "ERROR"
 60 GOTO 10
100 PRINT "RESULTADO";2+2
200 END
```

En esta ocasión, la línea 20 se encarga de extraer el carácter situado a la izquierda de la cadena introducida en B$.

TRATAMIENTO DE CADENAS EN EL BASIC SINCLAIR

Quizás los únicos ordenadores que presentan diferencias sustanciales en la forma de tratar a las cadenas alfanuméricas son los de la firma Sinclair (el popular ZX-SPECTRUM, por ejemplo). Estos equipos no hacen uso de ninguna de las funciones habituales (LEFT$, RIGHT$ y MID$), sino que recurren al empleo de una función propia. Esta es TO, cuyo formato es el siguiente:

<argumento>(N TO M)

Al igual que en el caso genérico, la zona de argumento incluye a la cadena sobre la que debe actuar la función. N señala el punto inicial o de "corte", a partir del cual se extraerán los caracteres. Por último, M indica el último carácter que forma parte de la subcadena a extraer. El ejemplo resuelto anteriormente por medio de la función MID$, adoptará ahora el siguiente aspecto:

```
 1 REM RUPTURA DE UNA CADENA
10 B$="123456789"
20 FOR I=1 TO 9
30 PRINT B$(I TO I)
40 NEXT I
■
```

Su ejecución conducirá a idéntico resultado:

```
RUN
1
2
3
4
5
6
7
8
9
```

Las funciones de extracción de caracteres por la izquierda y por la derecha de la cadena LEFT$ y (RIGHT$) son perfectamente operables con esta nueva sintaxis. Al omitir el valor N en la expresión general, el equipo extraerá los caracteres situados a la izquierda de la cadena, hasta llegar al que ocupa la orden M. En efecto, esta formulación equivale a la propia de la función LEFT$. Así, por ejemplo, la función LEFT$(A$, X) se representará en el BASIC de Sinclair de la siguiente forma: A$(TO X).

Algo semejante ocurre al omitir el valor final (M). Por ejemplo, la función A$(N TO), construirá una cadena constituida por los caracteres A$ situados a partir de la posición N hasta el final.

Existe aún otra salvedad, esta vez propia del BASIC Sinclair que incorpora el Sinclair-QL. En el texto se ha mostrado la imposibilidad de operar con datos de distinto tipo (numéricos y alfanuméricos) mezclados. El QL de Sinclair evita este problema, adecuando automáticamente el tipo del dato al de la variable que recibe la asignación. En el siguiente ejemplo:

```
A=17+"2"
```

La variable numérica A guardará la suma del dato numérico 17 y el dato numérico 2. Lo que hace es convertir previamente el carácter alfanumérico "2" a su correspondiente valor numérico (2). Así pues, además de no producir un mensaje de error, su contenido será el siguiente:

```
PRINT A <CR>
19
■
```

Logo (12)

Programación recursiva

Cualquier lista de instrucciones LOGO debe estar encabezada por un comando. Si lo que se desea es que esa lista dé como resultado un dato de salida, es evidente que éste debe de ser canalizado adecuadamente. En este segundo caso, la lista actúa como un operador.
Existe una orden LOGO especializada en "encauzar" datos de salida. La palabra LOGO que la identifica es OUTPUT.

```
TO RESTA :M :N
OUTPUT :M-:N
END
```

El procedimiento RESTA opera la sustracción en modo prefijo. En su interior, el comando OUTPUT es quien se encarga de que el resultado aparezca como dato de salida del procedimiento.
De este modo es posible construir procedimientos que actúen como operadores; claro está, estos procedimientos deben ir precedidos por un comando.
Además de permitir la creación de nuevos operadores, esta característica permite, también, hacer uso de un nuevo concepto: la recursividad. Este término, muy utilizado en teoría matemática, se refiere a la posibilidad de incluir lo definido dentro de la definición. Para ilustrar este nuevo concepto, construiremos un procedimiento cuyo cometido es definir los números pares de forma recursiva. El punto de partida se halla en la siguiente constatación:

El número N es par si:
- N es igual a 2, ó
- N−2 es par.

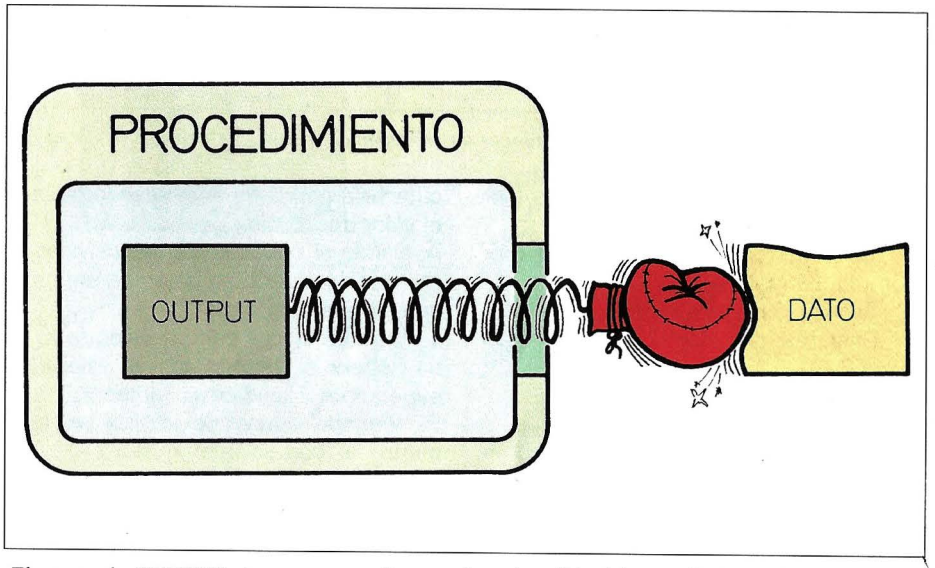
Sin lugar a dudas, este método para la detección de números pares resulta bastante engorroso para utilizarlo "a mano". No obstante, el ordenador puede ponerlo en práctica muy fácilmente:

```
TO PAR :NU
IF :NU=2 [OUTPUT "TRUE] [IF PAR :NU-2
[OUTPUT "TRUE]]
END
```

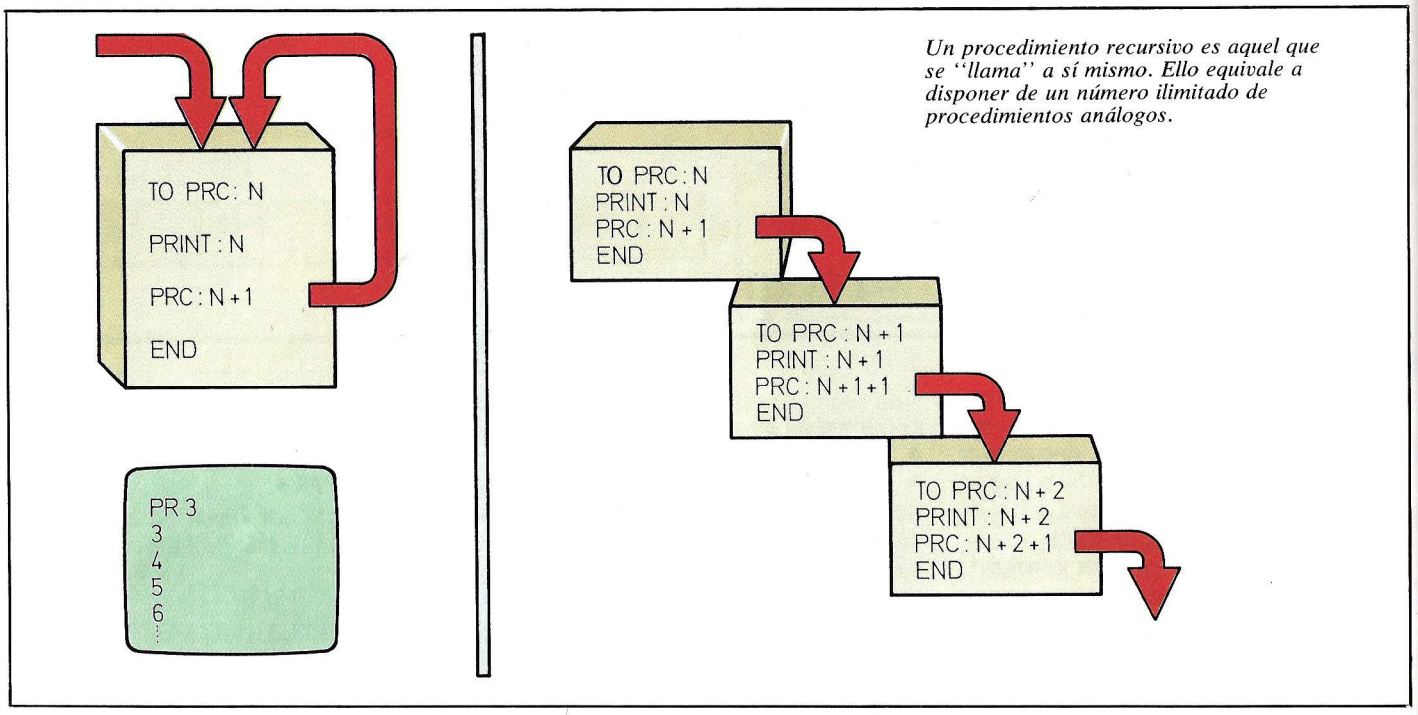
El procedimiento empieza comparando el dato de entrada con el número 2. Si no son iguales, se procede a efectuar la siguiente comparación; en esta segunda comparación se vuelve a hacer uso de PAR, aunque ahora el dato comparado con el 2 es N−2. En el supuesto de que el número N fuera 6, la ejecución coincidirá con la que se expone a continuación:

```
PAR 6
      IF 1)6≠2
      IF 2) PAR 4
                  IF 2.1)4≠2
                  IF 2.2) PAR 2
                              IF 2.21)2=2
                        TRUE
            TRUE
TRUE
```

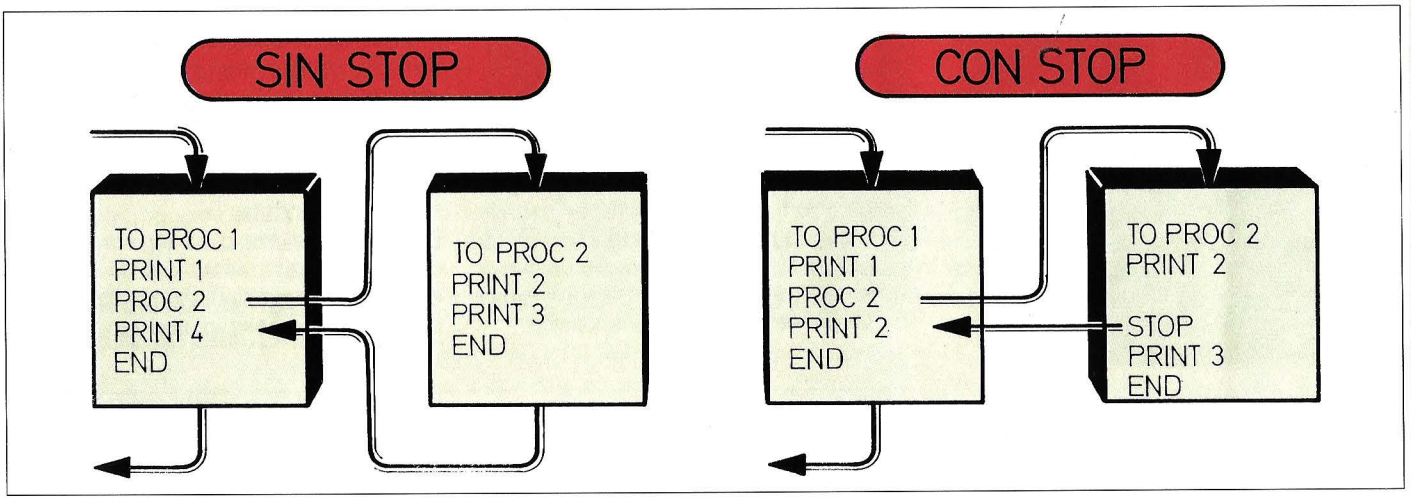
Tal como se observa, el procedimiento PAR se llama a sí mismo en dos ocasiones. Es posible que el ejemplo anterior resulte de ardua asimilación. Realmente, el concepto de recursividad exige un gran poder de abstracción. En todo caso, su compresión puede verse facilitada con la ayuda de la tortuga.
El siguiente ejemplo conjuga la recursividad con el tratamiento gráfico. Se trata, exactamente, de dibujar un polígono espiral. Ello puede conseguirse como sigue:



El comando OUTPUT sirve para canalizar un dato de salida del procedimiento. Dicho procedimiento se convierte, de esta forma, en un virtual operador LOGO.



Un procedimiento recursivo es aquel que se "llama" a sí mismo. Ello equivale a disponer de un número ilimitado de procedimientos análogos.



El uso del comando STOP permite abandonar un procedimiento, antes de que éste llegue al final del mismo delimitado por la orden END.

1. Avanzar la tortuga X posiciones (por ejemplo, 5).
2. Girar un determinado ángulo (por ejemplo, 90 grados).
3. Avanzar 2+X posiciones.
4. Girar el mismo ángulo señalado en el punto 2.
5. Avanzar 4+X posiciones.

...

Las acciones de "avanzar" y "girar" pueden agruparse en una lista. Por ejemplo:

[FORWARD :X LEFT 90]

La referida lista de instrucciones debe ejecutarse repetidamente alterando tan sólo el valor de :X.

Apelando al concepto de recursividad, el método apropiado se resumirá en los siguientes puntos:

a - Ejecutar la lista para un valor de X.

b - Repetir el método con un valor de X superior en 2 unidades al anterior.

El segundo enunciado implica volver al punto "a" con un valor igual a 2+X. Este ciclo se repetirá en sucesivas ocasiones, con valores crecientes de X en dos unidades por cada recorrido. La puesta en práctica de esta secuencia equivale a crear un polígono espiral en modo recursivo.

El procedimiento LOGO adecuado, presentará el siguiente aspecto:

```
TO ESPI :X
FORWARD :X
LEFT 90
ESPI 2+:X
END
```

El valor inicial de la variable X puede ser fijado a voluntad; de acuerdo a su magnitud se obtendrán espirales de distinto tamaño. Por supuesto, también es posible introducir el ángulo a modo de variable. E incluso puede escribirse una variante del anterior procedimiento que aporte una variación en el incremento sucesivo del valor de X:

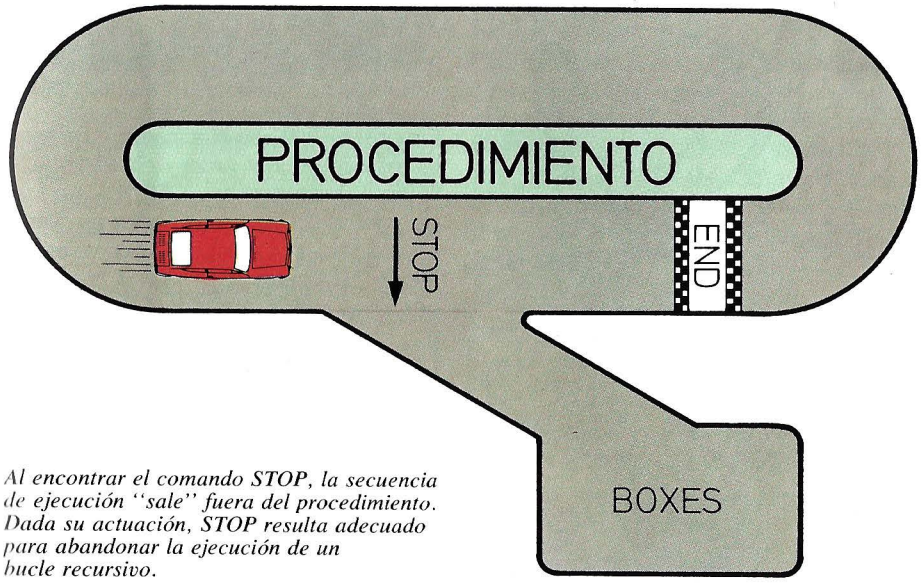
```
TO ESPIRA :X :A :I
FORWARD :X
LEFT :A
ESPIRA :X :A+I :I
END
```

Este último ejemplo permite las más variadas figuras, puesto que altera los valores iniciales de ángulo, longitud e incremento.

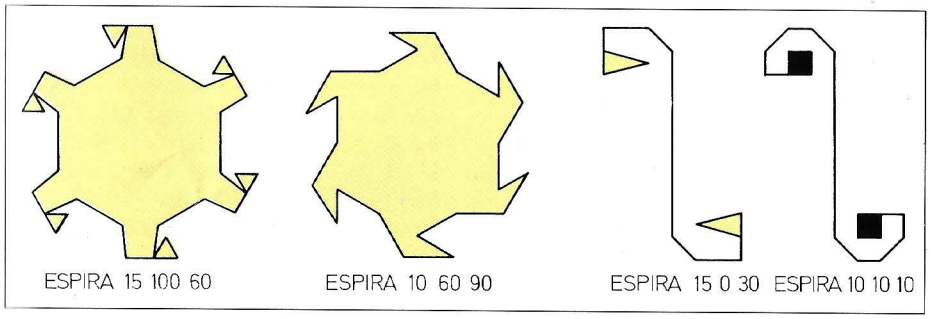
LIMITAR LA RECURSIVIDAD

En los ejemplos anteriores, la ejecución del procedimiento no tiene fin. El procedimiento continúa "llamándose" a sí mismo indefinidamente, y sólo se detendrá la ejecución al emitirse un mensaje de error: al exceder la variable del valor límite permitido. Desde luego, la interrupción también puede derivar de una orden al efecto dada por el propio usuario desde el teclado.

Normalmente, es necesario "limitar" la acción de un procedimiento recursivo. Ello es posible gracias al comando STOP. Su cometido se entenderá con mayor claridad dejando al margen, por el momento, el tema de la recursividad.



Al encontrar el comando STOP, la secuencia de ejecución "sale" fuera del procedimiento. Dada su actuación, STOP resulta adecuado para abandonar la ejecución de un bucle recursivo.



*Resultado de la ejecución del procedimiento **ESPIRA** con distintos parámetros de entrada.*

```
TO PROC1
PRINT 1
PROC2
PRINT 4
END

TO PROC2
PRINT 2
PRINT 3
END
```

El primer procedimiento (PROC1), tras escribir el dato 1, "llama" al segundo (PROC2). Este último mostrará en la pantalla los números 2 y 3; tras ello, devolverá el control a PROC1. Acto seguido, proseguirá la ejecución de PROC1 escribiendo la cifra 4.

Veamos, en definitiva, cuál es el resultado de su ejecución:

```
PROC1

1
2
3
4
```

Modifiquemos ahora, ligeramente, el subprocedimiento PROC2. Su nuevo aspecto es el siguiente:

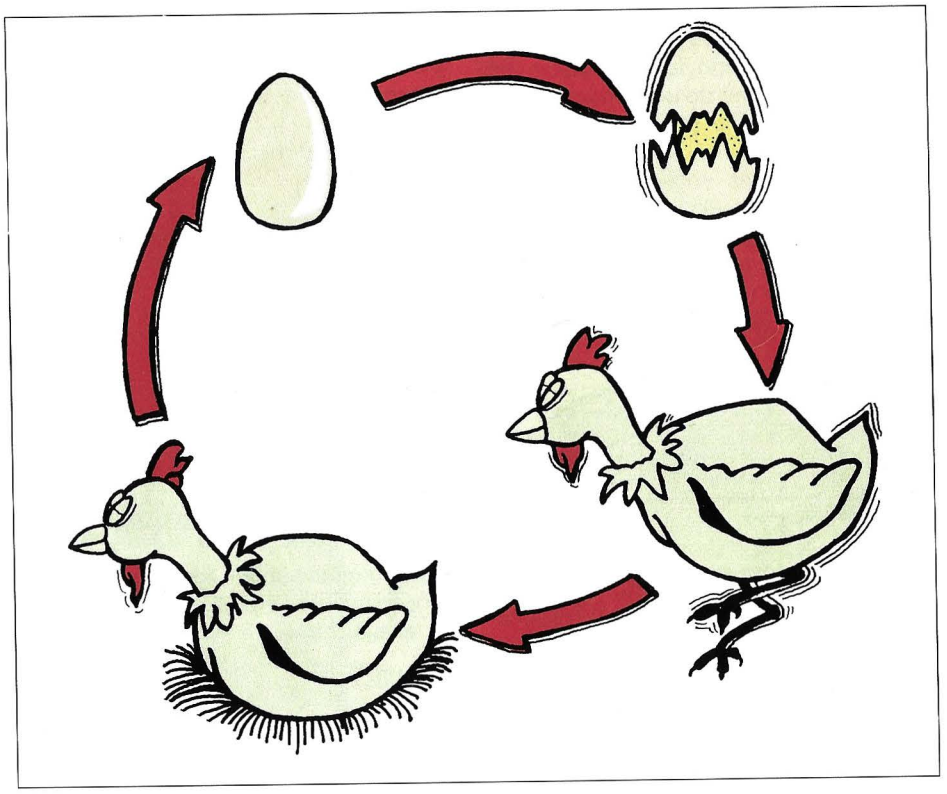
```
TO PROC2
PRINT 2
STOP
PRINT 3
END
```

La alteración queda plasmada por la presencia de un comando STOP entre las dos órdenes de escritura. Al ejecutar de nuevo el procedimiento PROC1, la pantalla reflejará el efecto de la modificación operada:

```
PROC1

1
2
4
```

TABLA DE ORDENES-LOGO

INSTRUCCION	COMETIDO	OPERADOR/ COMANDO
OUTPUT <dato>	Coloca su dato de entrada como dato de salida del procedimiento en el que se encuentra	Comando
STOP	Interrumpe la ejecución de un subprocedimiento para volver al procedimiento que lo llamó	Comando



La recursividad hace que la secuencia de ejecución entre en un círculo cerrado; una vez en su interior se diluye el significado de entrada y salida o principio y final del procedimiento... ¿Qué fue antes, el huevo o la gallina?

Como puede apreciarse, la instrucción PRINT 3 no se ha ejecutado. En cambio, sí aparece escrito el 4.

La acción de STOP es devolver el control al procedimiento que efectuó la llamada al procedimiento en el que se encuentra dicho comando. En otros términos: STOP retorna la ejecución al nivel inmediatamente superior (entendiendo que los procedimientos que "llaman" se sitúan en un nivel superior a los "llamados"). De no existir un nivel superior —el procedimiento no ha sido llamado por otro—, aparecerá el mensaje de error YOU'RE AT TOP LEVEL (está usted en el nivel superior).

Este comando va a resultar muy útil en procedimientos recursivos. Al "llamarse" a sí mismos, este tipo de procedimientos van recorriendo niveles cada vez inferiores.

En los ejemplos anteriores se descendía de nivel indefinidamente. Ahora, con la ayuda de STOP, se puede volver a subir hasta llegar al nivel de partida. He aquí un ejemplo sencillo en el que se conjuga la recursividad con el uso del comando STOP.

```
CUENTA :N
IF :N=0 [PRINT "CERO STOP]
PRINT :N
CUENTA :N-1
PRINT :N
END
```

El comando STOP actúa deteniendo el proceso recursivo. Una vez que se ha regresado al nivel inmediatamente superior, ya no se "bajará" de nuevo. Ello se debe a que, después de llamar al nivel inferior (cuarta línea), el resto del procedimiento no vuelve a efectuar una nueva llamada. El resultado de la ejecución adoptará el siguiente aspecto:

```
CUENTA 4
4
3
2
1
CERO
1
2
3
4
```

El primer PRINT es el encargado de escribir los números en orden descendente. Al llegar a 0, se escribe "CERO" y se interrumpe la recursividad. La sucesiva vuelta al nivel superior produce la cuenta ascendente, cuyos valores llegan a la pantalla por medio del segundo PRINT. La siguiente "radiografía" de la ejecución ilustra el desarrollo del proceso. Para no prolongar excesivamente el ejemplo, se ha tomado el número 2 como dato de partida.

línea	instrucción
1	CUENTA :N (2)
2	IF :N=0 ... (2 < > 0)
3	PRINT :N (2)
4	CUENTA :N − 1 (1)
1	CUENTA :N (1)
2	IF :N=0 ... (1 < > 0)
3	PRINT :N (1)
4	CUENTA :N − 1 (0)
1	CUENTA :N (0)
2	IF :N=0 [PRINT "CERO STOP]
5	PRINT :N (1)
6	END
5	PRINT :N (2)
6	END

ATARI. DOS II

Sistemas operativos en el ámbito de los pequeños equipos

La enorme difusión de la microinformática en los últimos años se debe, en gran medida, a la irrupción en el mercado de pequeños ordenadores, de precio moderado y orientados primordialmente al ámbito doméstico personal. Su presencia ha acercado la informática a toda una amplia diversidad de usuarios, movidos por un no menos dilatado abanico de motivaciones: juegos, toma de contacto con la informática, desarrollo de aplicaciones para uso personal, ...

Tan espectacular éxito tiene un componente básico en la economía de estos equipos. Al margen de la evidente influencia del progreso tecnológico, la moderación del coste de un ordenador doméstico deriva de la posibilidad de utilizar como dispositivos periféricos, algunos aparatos frecuentes en la mayor parte de los hogares; básicamente magnetófonos a casete y receptores de televisión. La configuración básica de este tipo de ordenadores, a los que se les ha dado en denominar ordenadores domésticos o nanoordenadores, suele estar constituida por un receptor de TV (unidad de visualización), un magnetófono a casetes (unidad de almacenamiento) y el propio ordenador.

La grabación y lectura de programas en cinta magnética de tipo casete, es un proceso lento y de escasa potencia. Ello ha motivado a los diversos fabricantes a lanzar al mercado periféricos de almacenamiento de acceso más rápido y versátil y de superior capacidad. Entre estos, el protagonismo corresponde a las unidades de disco flexible, unidades a las que se incorpora un sistema operativo que permita su explotación.

Dentro de este entorno se encuentra el sistema operativo ATARI.DOS (sistema operativo de disco de la firma ATARI), en sus distintas versiones, desarrollado para los modelos clásicos ATARI 400 y 800 y los nuevos 600XL y 800XL.

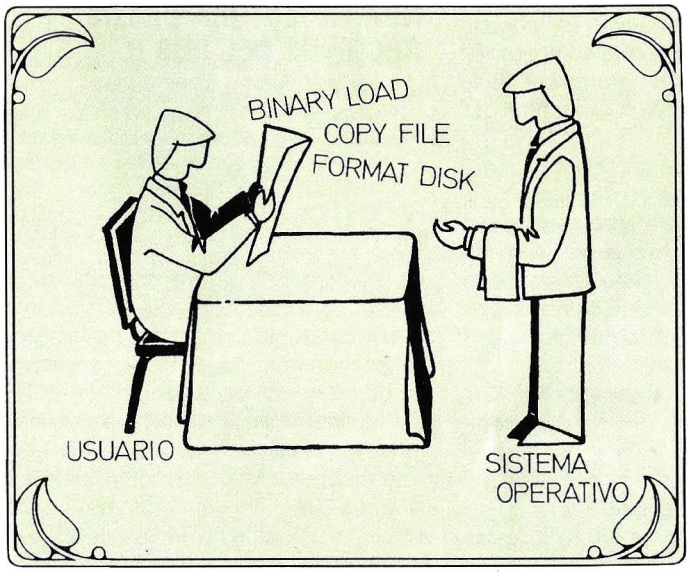
INTRODUCCION AL DOS II

El sistema operativo ATARI.DOS tiene dos versiones clásicas: el DOS I y el DOS II. Esta última corresponde a un desarrollo del DOS I que incluye varias mejoras e incorpora nuevos conceptos y opciones. De ahí que los próximos apartados se concreten en el estudio del ATARI.DOS II, la versión más completa de las dos señaladas.

Para tener acceso al empleo del sistema operativo DOS II, es preciso que el ordenador esté dotado de un mínimo de 16 Kbytes de memoria RAM: y, desde luego, que se encuentre asociado a una unidad para disco flexible del tipo ATARI 810 Disk Drive, ATARI 815 Dual Disk Drive o ATARI 1050 Disk Drive.

La principal característica del sistema operativo DOS II consiste en la incorporación de un menú de opciones, a través del cual se activan las diferentes funciones del sistema operativo. Para ello, basta sólo con elegir la letra que simboliza la acción a realizar y responder a ciertas preguntas formuladas por el sistema, con objeto de definir los parámetros necesarios. Esta peculiar forma de ejecutar los comandos, libera al usuario de la necesidad de memorizar una larga lista de comandos cuya sintaxis puede ser complicada. En su defecto, tan sólo debe elegir la acción a tomar de entre el repertorio de alternativas que refleja la pantalla.

El sistema operativo viene grabado en un disquete, denominado Disco Maestro. Su contenido incluye el paquete de gestión de archivos del sistema, así como todas las utilidades necesarias para el manejo de la unidad de disco. Cada disquete Maestro contiene los siguientes archivos:



La comunicación entre el usuario y el sistema operativo ATARI.DOS resulta cómoda y sencilla; ésta se establece con la simple selección de una de las funciones que brinda el menú.

- DOS.SYS

Este archivo contiene el subsistema para el manejo de archivos o FMS (File Management Subsystem) y la zona del archivo DUP.SYS que pasa a residir en RAM, conocida también como mini-DOS. Esta última porción del DUP.SYS contiene las subfunciones controladas por el subsistema de manejo de archivos (FMS): DELETE FILE, RENAME FILE, LOCK FILE, UNLOCK FILE y FORMAT DISK, cuyo significado se explicará más adelante.

- DUP.SYS

El archivo DUP.SYS memoriza el paquete de utilidades del disco o DUP (Disk Utylity Package), compuesto por el menú del sistema operativo y por las subfunciones que no están controladas por el FMS. En consecuencia, cuando se desee utilizar el menú del DOS o alguna de las funciones propias del DUP.SYS (BINARY LOAD, BINARY SAVE, RUN AT ADDRESS, RUN CARTRIDGE, COPY FILE, DUPLICATE FILE y DUPLICATE DISK), el contenido del archivo en cuestión debe ser transferido a la memoria RAM del sistema.

- AUTORUN.SYS

Se utiliza para verificar los dispositivos periféricos conectados al equipo, o para ejecutar programas en código máquina. Ambos son hechos que se producen de forma automática al inicializar el sistema operativo.

La carga del sistema operativo se efectúa automáticamente tras insertar el disquete Maestro en la unidad de disco definida como D1 (unidad principal). El proceso que sigue difiere según que en el preciso instante de la carga del sistema operativo se halle conectado o no un cartucho ROM en la ranura al efecto.

En el caso de que esté presente un cartucho, la carga se realiza en dos etapas:

a) Al introducir el disco Maestro, el archivo DOS.SYS pasa a la memoria RAM del equipo. El módulo en cuestión contiene el subsistema para el manejo de archivos FMS y el mini-DOS. En la pantalla aparecerá el mensaje propio del cartucho conectado en ese preciso instante (READY para el cartucho BASIC, EDIT para el editor, ...).

b) El paso siguiente es teclear DOS y accionar a continuación la tecla <RETURN>. Con ello, se producirá la carga del archivo DUP.SYS. Concluida ésta, la pantalla se verá ocupada por el menú del sistema operativo, listo para ejecutar el comando que elija el usuario.

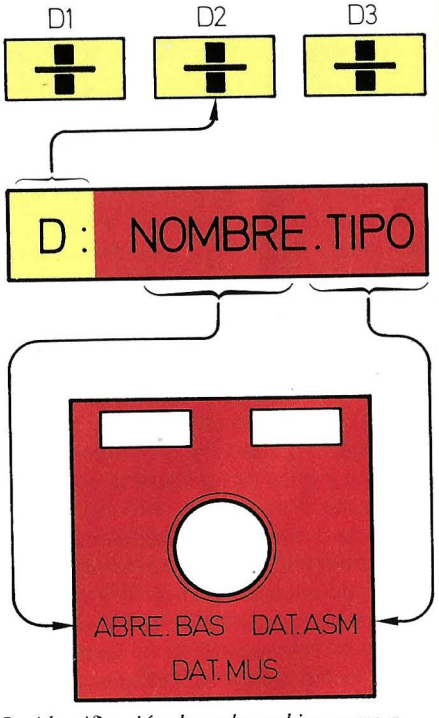
Por el contrario, de no existir ningún cartucho ROM conectado, la carga del sistema operativo se consuma tras introducir el disco Maestro en la correspondiente unidad. En tal caso, no es necesaria la intervención del usuario, cargándose automáticamente los ficheros DOS.SYS y DUP.SYS.

La inteligencia básica que aportan los sistemas operativos no queda reservada exclusivamente a los microordenadores y ordenadores personales de superior nivel. También los equipos domésticos suelen contar con sistemas operativos en disco que agilizan e incrementan su potencia y eficacia práctica. Una muestra elocuente se encuentra en el DOS II destinado a los ordenadores de la firma ATARI.

NOMENCLATURA DE LOS ARCHIVOS DEL DOS II

Antes de pasar a la descripción de los diversos puntos del menú, es preciso conocer cuáles son las reglas de formación de los nombres de los archivos a los que se van a referir las funciones del DOS II. Sin lugar a dudas, de igual forma que una persona no puede encontrar un cierto edificio en base a una dirección incorrecta, el sistema operativo será incapaz de encontrar en el disquete un archivo descrito con errores de sintaxis.

La nomenclatura de los archivos se rige



La identificación de cada archivo corre a cargo de una expresión en la que intervienen tres zonas: número del disco afectado, nombre otorgado al archivo y tipo del archivo en cuestión. En el ejemplo ilustrado, el campo de disco incluiría la referencia D2, puesto que alude a archivos almacenados en el disco insertado en la segunda unidad.

por una mínima cantidad de reglas simples, escuetas y capaces de determinar unívocamente la identidad de un archivo. Veamos cuáles son:

— Hay que indicar el código del dispositivo de almacenamiento de datos (disquete) y su número, en el caso de que exista más de una unidad conectada simultáneamente.

— El nombre del archivo debe separarse de la identificación del dispositivo por medio del signo dos puntos. El nombre no puede exceder de ocho caracteres; éstos pueden ser letras o números indistintamente, con la salvedad de que el primero de ellos debe ser necesariamente una letra.

— Para un mejor reconocimiento del archivo, es opcional la inclusión de tres números o letras, separadas del nombre por un punto, que referenciarán el tipo o contenido del archivo.

Atendiendo a las reglas indicadas, la denominación de un archivo será semejante a la que sigue:

D2 : NOMBRE. DAT: Archivo almacenado en la unidad de disco D2, denominada NOMBRE y cuyo contenido son datos.

En ciertos casos, los nombres de archivos pueden incluir dos signos especiales: el asterisco y la interrogación. En principio cabe imaginar que su sintaxis no es correcta, debido a la inclusión de caracteres no permitidos. Sin embargo, su presencia es correcta. El motivo obedece a que dichos caracteres se utilizan para definir "referencias ambiguas" o extensivas a dichos archivos. En efecto, es posible sustituir uno o varios caracteres por uno de estos signos. Concretamente, la interrogación sustituye a un único carácter, mientras que el asterisco hace lo propio con cualquier combinación de ellos. Por ejemplo:

D1 : PROY?.BAS: Alude a todos los archivos que estén en el disco D1, cuyo nombre contenga los caracteres PROY más otro cualquiera y cuyo contenido coincida con programas BASIC (tipo BAS).

D1 : *.DAT: Referencia a todos los archivos de datos (tipo DAT) almacenados en el disco 1, sea cual fuere su nombre.

EL MENU DEL DOS II

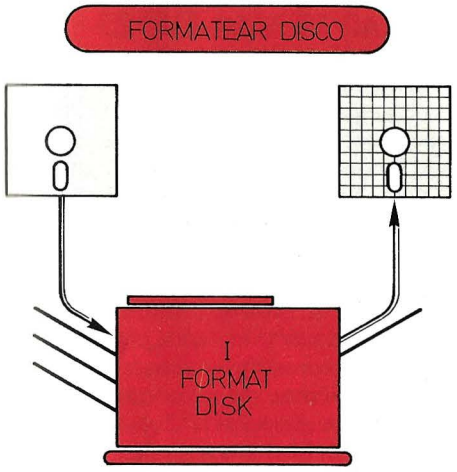
La aparición en pantalla de las diversas funciones que incorpora el sistema operativo, marca el instante a partir del cual el usuario tiene abierto el diálogo con el sistema. A partir de este momento, el sistema está preparado para recoger la opción del usuario y procesarla adecuadamente, incluso solicitando más información si ello fuera preciso. Acto seguido se describen las diversas acciones seleccionables.

A.DISK DIRECTORY

Muestra un listado de los archivos almacenados en el disco que se indique. El listado incluye el nombre de cada archivo, su tipo y el número de sectores que ocupa. La lista o directorio general puede ser total o parcial, dependiendo de los parámeros introducidos. Por ejemplo:

SELECT ITEM OR RETURN FOR MENU
A <RETURN>
DIRECTORY -SEARCH SPEC,LIST FILE?
*D1: * .DAT <RETURN>*
UNO DAT 038
DOS DAT 020
SELECT ITEM OR RETURN FOR MENU

El caso reflejado por el ejemplo genera el listado por pantalla de todos los archivos



El formateado o inicialización de los discos vírgenes, es una de las funciones contempladas en el menú general del sistema operativo ATARI.DOS II.

Diferencias entre los sistemas operativos DOS I y DOS II

El estudio acometido en el texto se concreta en la versión DOS II del sistema operativo en disco de ATARI. Su mayor amplitud y potencia, respecto al DOS I, justifica tal elección como base del estudio. No obstante, es posible que algún usuario disponga exclusivamente de la versión DOS I, con la que no es posible realizar algunas de las operaciones anteriormente descritas. Para diferenciar las capacidades específicas de cada una de ambas versiones, se detallan a continuación las diferencias más significativas antre DOS I y DOS II.

El método de formateo de los discos es la primera diferencia sustancial entre ambas versiones, hasta el punto de que los disquetes formateados bajo el control de DOS I no son compatibles con la versión DOS II. Ello se debe a que esta última versión necesita tres sectores para la carga del sistema (boot), permitiendo así el manejo de disquetes de doble densidad. A su vez, el DOS I sólo exige un sector para la carga del sistema y puede operar únicamente con disquetes de simple densidad.

Las principales carencias del DOS I con respecto al DOS II se resumen a continuación:

— No dispone de una utilidad que preserve el contenido de la memoria, anterior a la carga del sistema operativo, en un archivo (MEM.SAV).

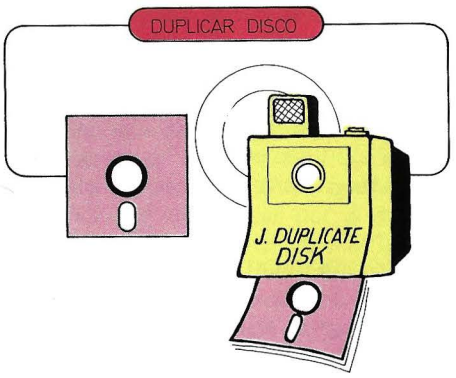
— No admite el uso de "referencias ambiguas" en la denominación de los ficheros.

— No incorpora un archivo AUTORUN.SYS que permita la ejecución automática.

— No dispone de los parámetros INIT y RUN, implicados en la carga en un archivo de una zona de la memoria.

— No muestra los sectores erróneos que pudieran detectarse al formatear un disquete.

Con respecto a algunas operaciones especiales, el DOS I es menos flexible o cómodo que el DOS II. Ello se evidencia en la necesidad de volver al menú general cada vez que termina la ejecución de un comando. No obstante, el DOS II permite optar entre volver al menú o ejecutar otro comando.



Una de las funciones disponibles en el menú del sistema operativo es la de duplicar discos (opción J). El ordenador realizará una copia íntegra, sector a sector, del contenido de un disco en otro.

D.DELETE FILE
Borra el archivo especificado.

E.RENAME FILE
Cambia el nombre de un archivo.

F.LOCK FILE
Bloquea un archivo de tal forma que quede protegido frente a operaciones de escritura; mientras se encuentra en dicho estado, el archivo tampoco puede ser borrado, cambiado de nombre o ampliado. Estos archivos se distinguen por la presencia de un asterisco delante del nombre al ordenar un listado del directorio.

G.UNLOCK FILE
Desbloquea un archivo y anula las características especiales que se detallan en el punto anterior.

El sistema operativo ATARI.DOS II reside en un disco flexible de 5 y 1/4 pulgadas. Este es ejecutable en las unidades de disco ATARI-810 Disk Drive (reflejada en la fotografía), ATARI-815 Dual Disk Drive o en el modelo más reciente y en distribución actual ATARI-1050 Disk Drive.

del disquete D1 que sean del tipo DAT. (Cabe señalar que los datos que aparecen en cursiva son los introducidos por el usuario).

B.RUN CARTRIDGE
Sirve para devolver el control al cartucho insertado o, directamente, al intérprete BASIC residente en la ROM interna.

C.COPY FILE
Copia el contenido de un archivo en el nuevo archivo indicado.

H.WRITE DOS FILE
Copia los archivos que contienen el sistema operativo (DOS.SYS y DUP.SYS) en un nuevo disco, previamente formateado.

I.FORMAT DISKETTE
Con esta función se organiza internamente el disquete, con objeto de que el sistema operativo sepa dónde puede emplazar la información. Hay que tener en cuenta que al ejecutar esta opción sobre un disco que contenga información, ésta quedará destruida por el proceso de formateo.

J.DUPLICATE DISK
Se utiliza para copiar íntegramente un disco en otro. La copia se realiza sector a sector, de manera que el disquete copiado es un fiel reflejo del original. Cuando en este proceso sólo se dispone de una unidad de disco, el sistema utiliza la zona de memoria RAM donde residen los programas BASIC como almacenamiento intermedio, perdiéndose, por lo tanto, el contenido de esta porción de memoria.

K.BINARY SAVE
Esta función permite grabar información en formato binario, semejante al obtenido cuando se emplea la unidad editora del ensamblador.

L.BINARY LOAD
Adecuado para cargar el contenido de un archivo almacenado mediante la opción precedente (grabación binaria).

M.RUN AT ADDRESS
Cuando un archivo en formato binario ha sido cargado en memoria y en él no aparece especificada la dirección de arranque del programa, es preciso indicar tal dirección por medio de la opción M.

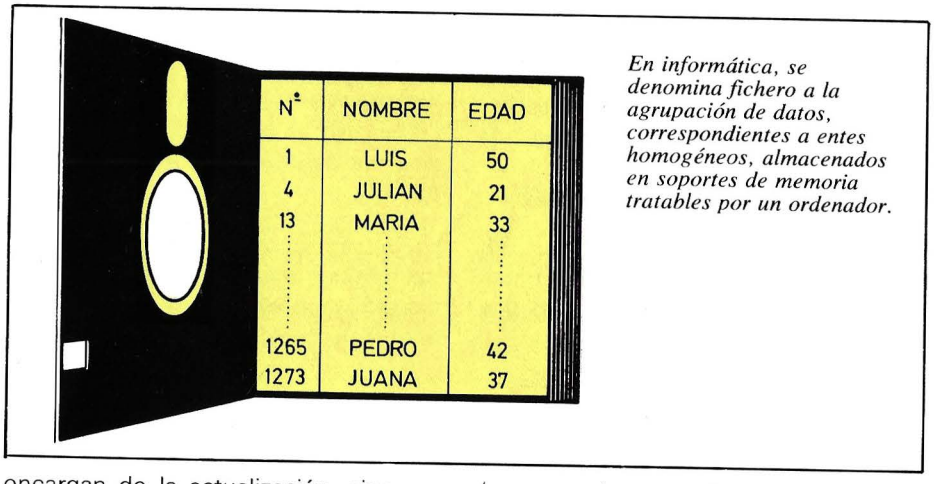
N.CREATE MEM.SAV
En el caso de que al invocar al sistema operativo existiese un cartucho insertado, el contenido de la memoria puede llegar a perderse al producirse la carga del mismo. Para evitar esta circunstancia, el usuario cuenta con la opción N. Esta crea un archivo (MEM.SAV) al que se traslada el contenido de la memoria antes de proceder a la carga del sistema operativo. Una vez concluido el trabajo con el sistema operativo, el archivo MEM.SAV devuelve automáticamente su contenido a la memoria del equipo, restaurándose así la información existente antes de la carga del sistema operativo.

O.DUPLICATE FILE
Opción adecuada para duplicar archivos de una a otra unidad de disco. Esta operación puede afectar al contenido del archivo MEM.SAV si dicha copia se realiza con una sola unidad de disquete.

Orden en los datos

Sistemas para la gestión de bases de datos

Tradicionalmente, en Informática, se ha denominado *fichero* a un conjunto de datos homogéneos, y *banco de datos* a toda colección de distintos ficheros dedicados a una misma aplicación.
Hace apenas media década, apareció un nuevo producto software al que se denominó *base de datos*. Muchos usuarios comenzaron a utilizar esta expresión para designar indistintamente a ficheros y bancos de datos. No cabe duda, pues, que antes de entrar en materia, es preciso clarificar el significado real de base de datos.



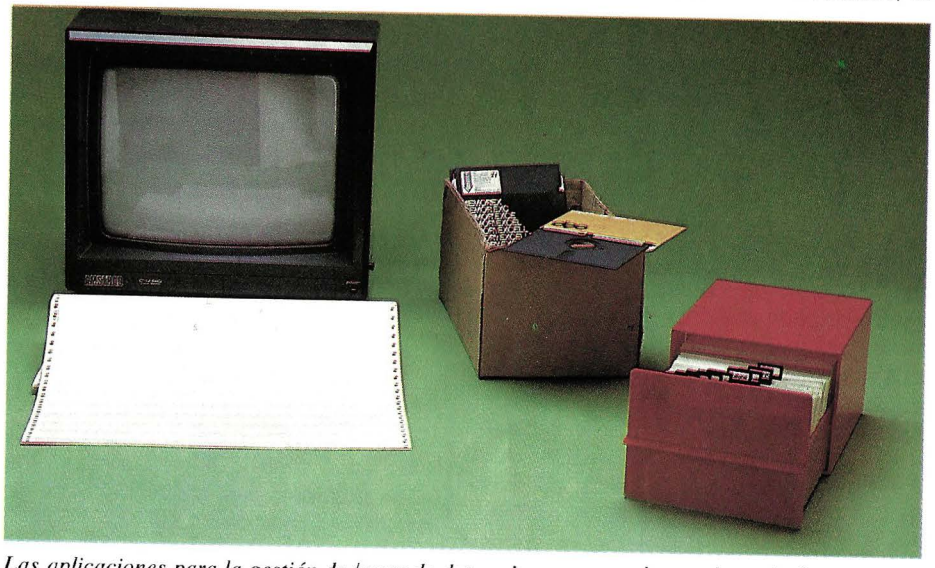
En informática, se denomina fichero a la agrupación de datos, correspondientes a entes homogéneos, almacenados en soportes de memoria tratables por un ordenador.

¿QUE ES UNA BASE DE DATOS?

En primera instancia, cabe afirmar rotundamente que una base de datos no es un fichero y tampoco un banco de datos. Una definición clásica de base de datos puede ser la siguiente: una base de datos es un conjunto de datos relacionados entre sí que se encuentran almacenados en una única colección, sin redundancias innecesarias, y que cumple las siguientes condiciones:
1. Los datos se encuentran almacenados en diversos soportes de información, de tal forma que son independientes de los programas que los manejan.
2. Su utilización no está restringida a una aplicación exclusiva, siendo posible su acceso por parte de varias aplicaciones, incluso simultáneamente.
3. Para gestionar la información contenida en la base de datos —esto es: para incluir nuevos datos, borrar datos ya existentes, o modificarlos—, se emplean procedimientos especialmente diseñados para optimizar el funcionamiento del sistema. Estos procedimientos no sólo se encargan de la actualización, sino que también permiten la obtención de datos para su uso o mediante programas.

Al resumir la anterior definición formal, destacan dos conceptos fundamentales:
1. Una base de datos es información almacenada en soportes.
2. Una base de datos son procedimientos (programas) encargados de gestionar información.

Esta doble faceta es la que ha dado lugar a innumerables confusiones. Por un lado, algunos asocian base de datos con información, mientras que otros asocian dicho concepto con los programas para la gestión de esta información. En realidad, el



Las aplicaciones para la gestión de bases de datos vienen a reemplazar a los métodos manuales para el archivo y la extracción organizada de informaciones.

concepto de base de datos integra ambas características.

BASES DE DATOS PARA MICROORDENADORES

La anterior definición de base de datos permite intuir que este producto ofrece grandes prestaciones y, en consecuencia, que los recursos informáticos (la máquina) en los que pueden ser explotados deben ser muy potentes. Este no es el caso de los microordenadores. De ahí que los paquetes para la gestión de base de datos en pequeños equipos, suelan simplificar su cometido hasta reducirlo, en algunos casos, a sencillos pero eficientes sistemas destinados a la gestión y mantenimiento de ficheros.

En general, las bases de datos para microordenadores permiten al usuario definir unos formatos de pantalla en los que se introducirá la información, y otros formatos de pantalla o impresora que se utilizarán para extraer información de la base. Los primeros suelen limitarse a un grupo de literales, descriptivos de los datos a introducir, y a unos "campos variables" destinados a recibir la información. En cambio, los formatos de salida, además de incluir los dos tipos de atributos anteriores, suelen permitir la definición de datos calculados. Ello significa que, en un "campo variable", es posible especificar una fórmula aritmética o lógica; fórmula que servirá para que el sistema gestor de la base obtenga un resultado concreto en función de otros datos.

Otra característica muy importante de las bases de datos es su versatilidad para obtener información: una vez definido el formato en el que se desea obtener el informe, se pueden definir condiciones para la inclusión de elementos.

Veamos un ejemplo. Suponga que disponemos de una base de datos con información sobre los clientes de una empresa y un formato de salida por pantalla o impresora; en este último aparece el número del cliente, su nombre, la provincia en la que está domiciliado, el tipo de producto distribuido y el volumen total de ventas. En tal caso, es posible obtener tres tipos distintos de informes con el mismo formato:

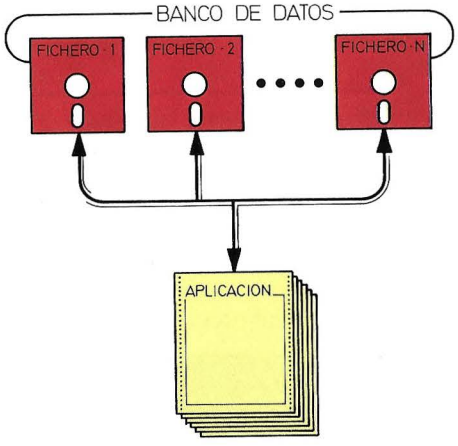
1. *Informes completos.*
Sin más que solicitarlo, el sistema proporcionará un listado con la información incluida en el formato para todos los clientes de la empresa.

2. *Informes parciales.*
Imponiendo alguna condición, por ejemplo que el volumen de ventas sea superior a dos millones de pesetas, se obtendrá un listado con toda la información del formato, pero sólo para los clientes que satisfagan la condición impuesta.

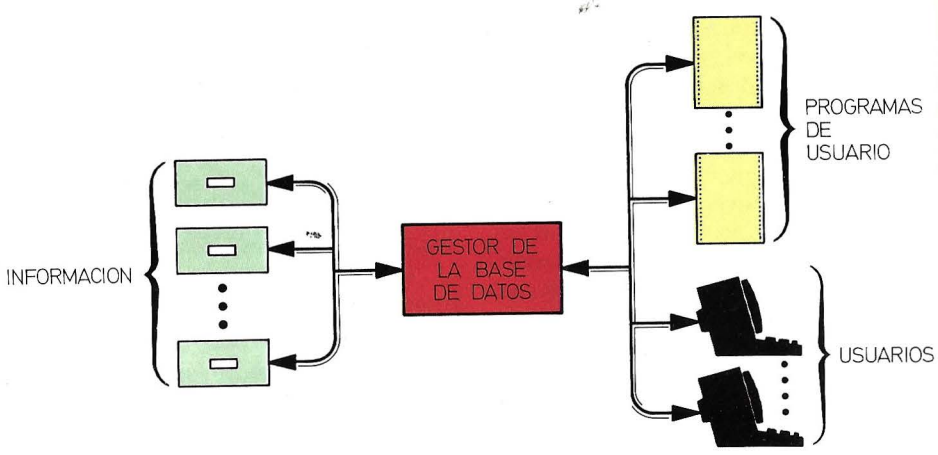
3. *Informes elementales.*
Limitados a un único cliente; aunque, por supuesto, ajustándose los resultados al formato definido. En el fondo, este tipo de informe no es más que una restricción del anterior, ya que la forma de obtenerlo se reduce a imponer una condición de gran exclusividad; por ejemplo, que el número del cliente sea igual a 17.

La obtención de informes de esta índole, lleva asociada la imposición de condiciones que, en esencia, constituyen una forma de diálogo entre el usuario y la máquina. Este diálogo debe ser lo más sencillo posible. El caso óptimo sería comunicarse con el ordenador en lenguaje natural, por ejemplo, en castellano. Aunque ello parezca una utopía, no hay que olvidar que en la actualidad existen proyectos de investigación muy avanzados en este sentido. Como contrapunto, el caso más desfavorable consiste en utilizar un lenguaje tradicional de programación. Entre ambos extremos existen alternativas aceptables. Básicamente, cabe hablar de dos tipos de lenguajes para la comunicación con una base de datos.

1. Lenguajes basados en el cálculo de predicados, que permiten imponer condiciones mediante la utilización de operadores lógicos como AND (y), OR (o), o NOT (no) y operadores relacionales como EQ (igual a), GT (mayor estricto que), LT (menor estricto que), etc.
2. Lenguajes basados en ejemplos, que permiten al usuario interrogar a la base de datos sin más que mostrar un sencillo ejemplo del informe que esperan obtener. Evidentemente, los lenguajes basados en



Cuando un paquete de programas o aplicación trabaja simultáneamente con una colección de ficheros, tal agrupación recibe el nombre de banco de datos.

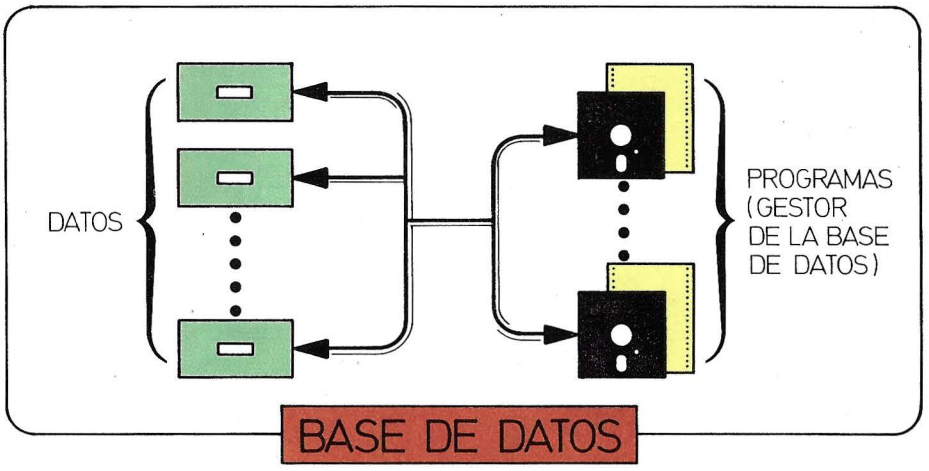


El objetivo fundamental de la base de datos es permitir un tratamiento diferenciado de la información almacenada, por parte de los diversos programas y usuarios que la utilicen.

el cálculo de predicados son más potentes que los basados en ejemplos, aunque, a cambio, también son más complicados de utilizar.

REDUNDANCIA E INCONSISTENCIA

Antes de dar por terminada esta introducción a las características generales de los sistemas para la gestión de bases de datos, hay que analizar dos factores que pueden dar al traste con las ventajas que ofrecen estos paquetes de aplicación: la *redundancia* y la *inconsistencia*.
En las bases de datos para grandes ordenadores, suele haber mecanismos de protección para obviarlos. En cambio, en las aplicaciones desarrolladas para ordenadores personales, debe ser el propio usuario quien garantice su ausencia.



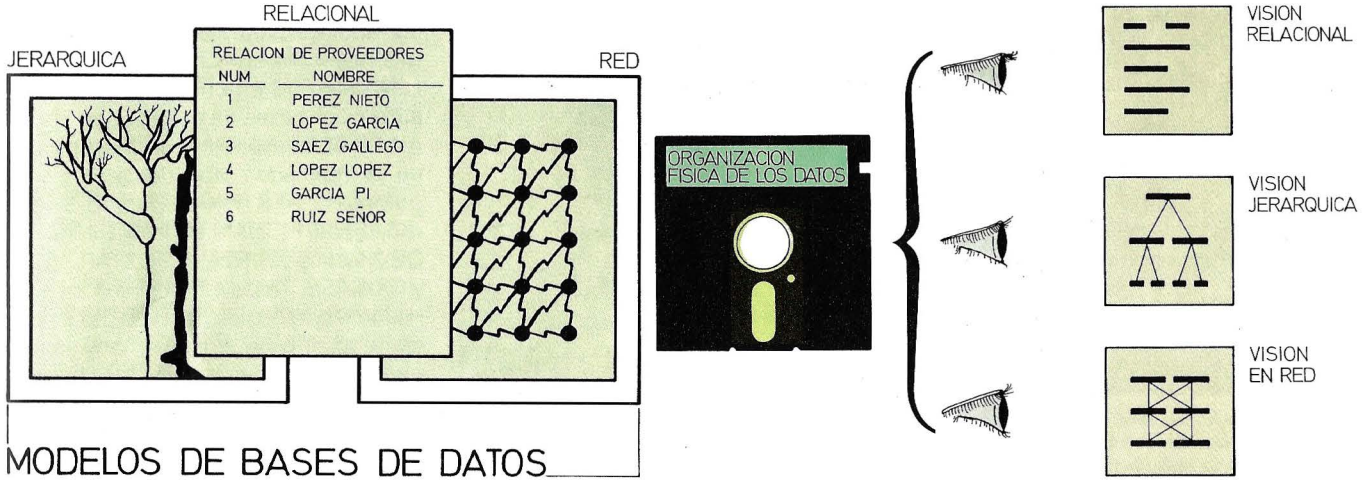
Los dos elementos fundamentales de una base de datos son los propios datos memorizados y los programas destinados a la gestión de los mismos.

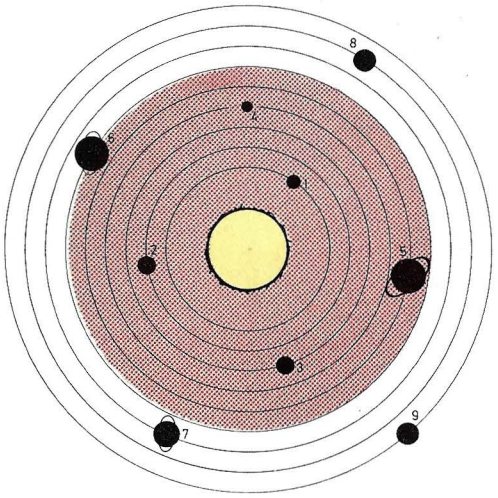
- *Redundancia*

Se define a la redundancia como el almacenamiento de un mismo dato en más de una posición del soporte físico. Esta duplicidad puede aparecer en ficheros distintos, o incluso en un mismo fichero. Por ejemplo, si para numerar a los clientes de una empresa se recurre al código de la provincia en la que habitan, seguido por un número de orden, y en otra posición distinta del mismo fichero aparece grabado el código de la provincia, resulta evi-

Modelos de bases de datos

Los nombres de los distintos modelos de bases de datos no pueden ser más elocuentes: RELACIONAL, JERARQUICA (o ARBORESCENTE) y RED. Está claro que la primera implica la posibilidad de trabajar con relaciones; tal es el caso de una sencilla relación de proveedores de una empresa, integrada por un número de orden y el nombre del proveedor.
En las bases de datos jerárquicas, la relación existente entre los datos será similar a la que hay entre los nodos de un árbol: la raíz se puede considerar como el nodo más importante jerárquicamente, debajo de ellas se encontrará el nodo terminal del tronco, etc. Por último, el modelo de red permite establecer una relación mucho más dinámica, ya que cada "nodo" de la red, que será el elemento fundamental, puede estar conectado con prácticamente cualquier otro "nodo".
De la anterior exposición, se puede deducir que la única diferencia existente entre los tres modelos, estriba en la forma en que permiten "visualizar" la información.
El término "visualizar" hace referencia tanto a la forma en la que pueden leer o escribir información en la base de datos los programas de aplicación, como a la estructura con la que puede trabajar directamente el usuario sin necesidad de que intervenga más que el propio software del sistema gestor de la base. Por lo tanto, es obvio que la diferencia entre dos modelos no estriba en la forma que adopta el almacenamiento físico de los datos en los soportes de información, sino en su método lógico de explotación.





1 MERCURIO	57,9	0,3794	✸
2 VENUS	108,1	0,97	
3 TIERRA	149,6	1	
4 MARTE	227,8	0,53	
5 JUPITER	778,2	11,19	
6 SATURNO	1.423,1	9,47	✸
7 URANO	2.878,3	3,73	✸
8 NEPTUNO	4.520	3,50	✸
9 PLUTON	5.928,1	0,468	✸

DISTANCIA DEL SOL (millones de Km.)
DIAMETRO (ø Tierra = 1)
PLANETAS FRIOS

Para ilustrar el concepto de "inconsistencia" podemos recurrir al ejemplo de la figura. Dentro del sistema solar, cabe considerar que son planetas fríos los que distan más de mil millones de kilómetros del Sol.
En base a este criterio, el fichero de planetas que se acompaña presenta una clara inconsistencia. En efecto: "o Mercurio no es un planeta frío, o dista bastantes más kilómetros del Sol".

dente que este código está produciendo redundancia.
No cabe duda que la redundancia es negativa. Por un lado, se derrocha espacio de memoria y, por otro, cada vez que haya que modificar el contenido del dato, será preciso acceder a más de una posición.
El método idóneo para eliminar la redundancia consiste en hacer un análisis de todos los datos a manejar y, en función de dicho análisis, agruparlos convenientemente.

- *Inconsistencia*

Se define a la inconsistencia como la generación de situaciones contradictorias debido a la existencia de datos almacenados que se contradicen. Suponga, por ejemplo, que en el mismo fichero de clientes del caso anterior, se modifica el código de provincia de uno de ellos, pero no se modifica el número de orden. El resultado dará lugar a una inconsistencia, ya que el contenido de la primera parte del número de orden guardará información de una provincia, mientras que el código de provincia tendrá un valor distinto.
En el ejemplo anterior, la inconsistencia es trivial. No obstante, en los casos reales se suele presentar más escondida y, normalmente, se muestra de forma transitiva en casos que no son fáciles de localizar.
El método adecuado para eliminar la inconsistencia se concreta en garantizar la no redundancia de la información; en la mayoría de los casos, ésta da origen a las situaciones de inconsistencia.

ALGUNAS BASES DE DATOS PARA MICROORDENADORES

Desde luego, la forma ideal para comunicarse con una base de datos es "a viva voz"; aunque ello no es lo habitual. No obstante, los modernos paquetes para la gestión de bases de datos, brindan al usuario sencillos lenguajes de comunicación que facilitan un diálogo canalizado a través del teclado y de la pantalla del ordenador.

Antes de citar a los principales sistemas para la gestión de bases de datos en microordenadores, hay que apuntar una precisión. En algunos casos, estos paquetes de aplicación no son auténticas bases de datos, sino que habría que denominarlos, más propiamente, "gestores de ficheros con índices para facilitar el acceso a los mismos". Sin embargo, también hay que constatar que en otros casos se trata de auténticas bases de datos. Entre los sistemas más relevantes y de mayor impacto en el mercado, cabe destacar a los siguientes: PFS File-Report, d-BASE II, d-BASE III, FRIDAY, OMNIS, DATA-STAR, DB-Master, VISIFILE, INFOSTAR, VISIDEX y CX-BASE. Todos ellos destinados a ordenadores personales para uso profesional o de gestión, en el ámbito empresarial. Hay que señalar que los microordenadores para uso doméstico (los denominados "nanoordenadores"), cuentan también con un amplio surtido de gestores para ficheros de información.

El ordenador como herramienta

La solución a tareas repetitivas

Una duda inmediata y compartida por los potenciales usuarios de un ordenador, es la que toma cuerpo en la siguiente pregunta: ¿de qué forma puede colaborar el ordenador en mi trabajo? La respuesta es sumamente compleja por lo extensa que puede resultar. Son tantas y tan variadas las tareas en las que tiene cabida el ordenador, que éste se convierte poco menos que en una máquina universal. Sin embargo, no hay que olvidar que, en definitiva, el ordenador no es más que una herramienta; un simple convidado de piedra si no cuenta con la instrucción del usuario.

Una sierra mecánica es capaz de abatir los árboles más poderosos, pero resulta inútil sin el hábil manejo de un experto talador. De la misma forma, un ordenador, sin la mano de un programador, se convierte en un simple amasijo de cables.

Las tareas para las que está más capacitado un ordenador son aquellas que implican un gran volumen de operaciones repetitivas. Esta es una afirmación cuya elocuencia queda plasmada en el siguiente ejemplo.

El problema se localiza en una compañía de alquiler de vehículos. Las tarifas se establecen con arreglo a dos posibles fórmulas:

a) 8.000 pesetas diarias sin límite de kilómetros.

b) 3.000 pesetas diarias más un suplemento de 15 pesetas por kilómetro recorrido.

La eficaz herramienta que es el ordenador, debe ser capaz de construir una tabla comparativa del coste de cada opción para distintos kilometrajes.

Denominando K a la variable que ha de contener el número de kilómetros recorridos, la cantidad a pagar de acuerdo a la segunda fórmula es:

```
P2=3000+K*15
```

Acogiéndose a la primera fórmula, la cantidad es siempre 8.000 pesetas, con independencia de los kilómetros recorridos.

En estas condiciones, se decide analizar los distintos costes variando el número de kilómetros. Tal variación comprenderá desde 50 a 500 kilómetros, con incrementos sucesivos de 25 kilómetros. Este es un cálculo repetitivo perfectamente encuadrable en un bucle del tipo FOR/NEXT:

```
FOR K=50 TO 500 STEP 25
```

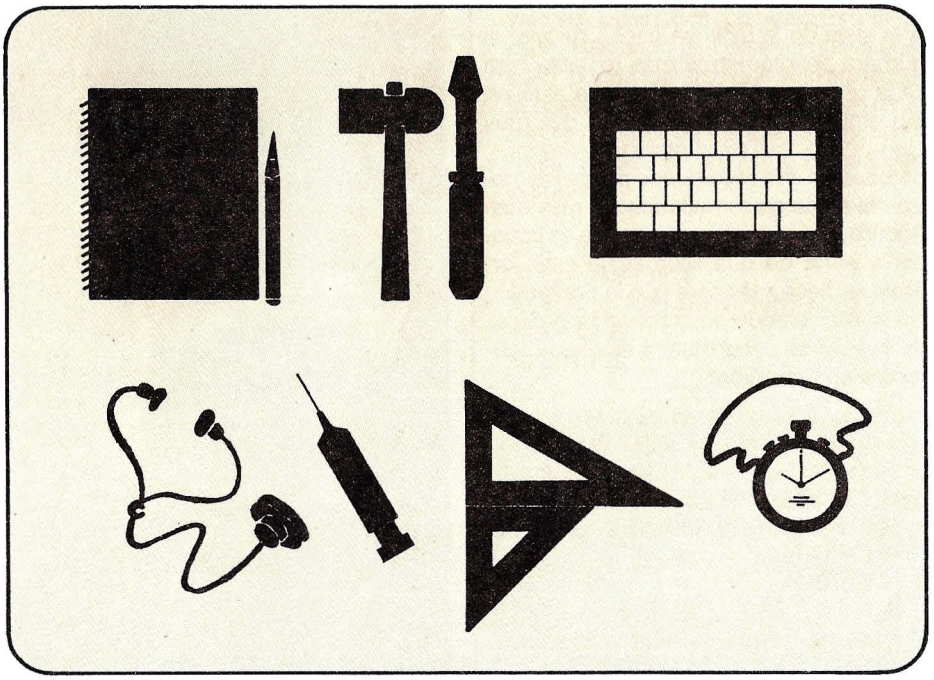
En la instrucción de apertura, 50 es el valor inicial, 500 el valor final y 25 el incremento para cada ejecución del bucle.

Una vez agrupadas las instrucciones oportunas, se llega al siguiente programa:

```
 1 REM ESTUDIO DE FORMULA DE ALQUILER
10 PRINT "KILOMETROS", "FORMULA1",
   "FORMULA2"
20 FOR K=50 TO 500 STEP 25
30 PRINT K, 8000, 3000+15*K
40 NEXT K
50 END
```

La línea 30 escribirá en primer lugar el valor de la variable K, seguido por el valor constante 8000 y, por último, el resultado de calcular 3000+15*K.

Si debido a la especial versión de su traductor BASIC, el ordenador no saltara los



El ordenador es una eficaz herramienta al servicio del usuario. Una herramienta capaz de colaborar en las más diversas tareas; por supuesto, siempre y cuando reciba la programación adecuada.

espacios adecuados al uso de las comas en la instrucción PRINT, puede sustituirse la línea 30 por la siguiente:

```
30 PRINT K;" ";8000;" ";3000+15*K
```

Y la 10 por:

```
10 PRINT "KM. -FOR.1 -FOR.2"
```

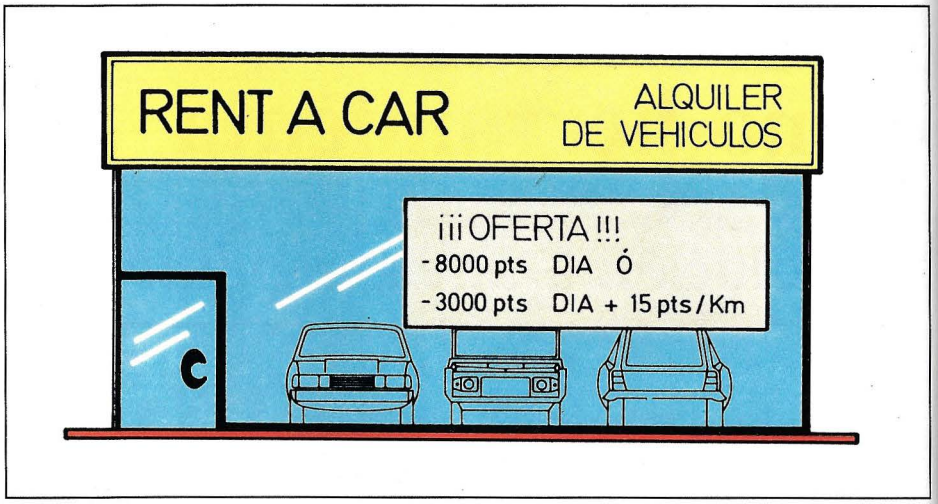
Tras ejecutar el programa, aparecerá en la pantalla el siguiente resultado.

```
RUN <CR>
```

KILOMETROS	FORMULA1	FORMULA2
50	8000	3750
75	8000	4125
100	8000	4500
125	8000	4875
150	8000	5250
175	8000	5625
200	8000	6000
225	8000	6375
250	8000	6750
275	8000	7125
300	8000	7500
325	8000	7875
350	8000	8250
375	8000	8625
400	8000	9000
425	8000	9375
450	8000	9750
475	8000	10125
500	8000	10500

A la vista de la tabla se puede calcular el número de kilómetros más rentable para cada una de ambas fórmulas de alquiler. Sin embargo, ésta es una tarea que muy bien puede realizar el propio ordenador. Se observa que recorriendo pocos kilómetros la segunda fórmula resulta más económica. Así pues, lo que interesa es conocer a partir de qué kilometraje esta fórmula se hace más cara que la primera. Un primer bloque de instrucciones capaz de evaluar tal circunstancia es el que aparece a continuación:

```
10 K=50
20 P2=3000+15*K
30 IF P2>8000 THEN PRINT K
40 K=K+25
50 GOTO 20
■
```



El ordenador puede demostrar ampliamente su eficacia en actividades que exijan un gran volumen de operaciones repetitivas. Un ejemplo práctico lo aporta el ejemplo desarrollado en el texto. En él se realiza el cálculo comparativo del coste asociado a cada una de ambas fórmulas de alquiler.

El ordenador: una eficaz herramienta o un simple amasijo de cables, plástico y componentes electrónicos; la diferencia reside en la mano del programador.

¿De qué forma puede colaborar el ordenador en mi trabajo? La amplitud de la respuesta tiene su límite en la capacidad del programador para "instruir" a la máquina.

En el mismo, se van incrementando los kilómetros en la línea 40, de 25 en 25 unidades. El número de kilómetros inicial se fija en 50 dentro de la línea 10. Tal como está planteado el programa, éste escribirá todos los valores de K para los que el importe según la fórmula 2 supera al precio que establece la opción 1.

Ha llegado el momento de introducir una nueva posibilidad habitual en el BASIC. Esta es la de incluir más de una instrucción por línea de programa. En efecto, en una misma línea pueden coexistir varias instrucciones. Para ello, hay que separarlas entre sí por medio del signo "dos puntos" (:).

Ahora, las dos últimas líneas del ejemplo pueden agruparse en una sola:

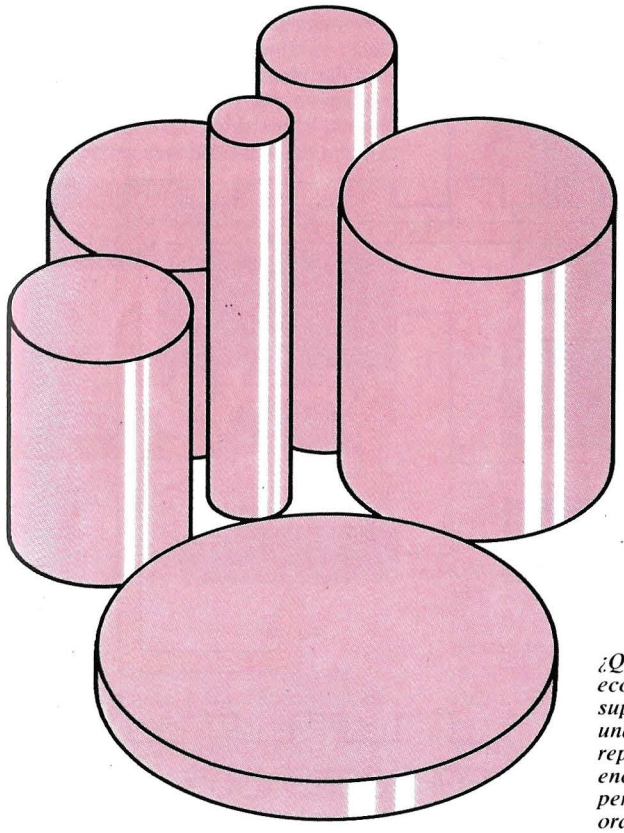
```
40 K=K+25 : GOTO 20
```

Ello supone un ahorro en líneas de programa. En todo caso, el mayor interés de la coexistencia de varias instrucciones en una misma línea, radica en las instrucciones del tipo IF/THEN. En el ejemplo anterior, se escribían en la pantalla todos los valores localizados por encima del buscado. Haciendo uso de esta nueva posibilidad, puede lograrse que sólo aparezca en la pantalla el valor crítico: número de kilómetros a partir de los que la segunda fórmula de alquiler resulta más cara que la primera.

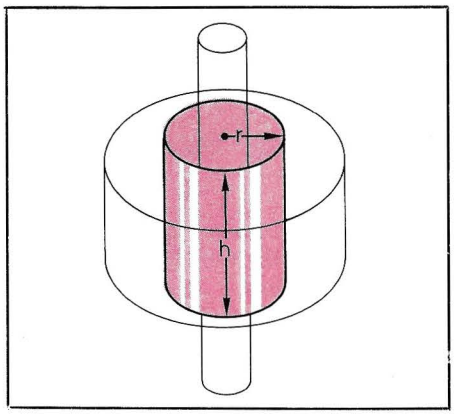
```
10 K=50
20 P2=3000+15*K
30 IF P2 > 8000 THEN PRINT K:STOP
40 K=K+25
50 GOTO 20
```

Una vez que se determina el valor en cuestión, se detiene la ejecución del programa. También es posible definir una presentación más atractiva, mostrando los kilómetros y el precio.

```
10 K=50
20 P2=3000+15*K
30 IF P2 > 8000 THEN PRINT K;" KM A ";
   P2;"PTAS":STOP
40 K=K+25
50 GOTO 20
```



¿Qué formato es el más económico? La decisión supone acometer toda una serie de operaciones repetitivas que pueden encomendarse perfectamente al ordenador.



Para llegar a establecer qué dimensiones son las óptimas, hay que variar en los sucesivos cálculos las medidas del radio (R) y de la altura (H) del bote.

Otra alternativa consiste en invertir la condición impuesta en la instrucción IF. Si bien, para no alterar el resultado, habrá que intercambiar las instrucciones de la zona THEN con las incluidas en las líneas 40 y 50. Este será el nuevo aspecto del programa:

```
10 K=50
20 P2=3000+15*K
30 IF P2 < 8000 THEN K=K+25:GOTO 20
40 PRINT K;" KM A ";P2;" PTAS"
50 STOP
```

En cualquiera de los dos últimos programas puede precisarse con mayor exactitud el resultado, cambiando el valor del incremento de 25 a 1 Km. Para ello, bastará con sustituir la asignación K=K+25

La intimidad del microprocesador

En el interior de un ordenador habita una densa amalgama de componentes electrónicos que son los responsables de su correcto funcionamiento; componentes que conforman los circuitos que dan vida al ordenador. Entre ellos destaca una zona fundamental, denominada *unidad central de proceso,* la CPU (Central Processing Unit). Ella es la encargada de realizar las tareas que implican una cierta "inteligencia". Un ordenador consta de otras zonas, si bien, la fundamental es la CPU: su centro neurálgico.

En los ordenadores personales y, en general, en cualquier microordenador, la CPU está condensada en un circuito integrado denominado microprocesador.

Los dos cometidos principales de esta unidad son el control del sistema conjunto y la realización de operaciones. Dentro del primer cometido se encuadran las tareas de organización y gestión de la memoria y los dispositivos periféricos. Sólo la detección y reconocimiento de la pulsación de una tecla implica ya un proceso complejo, en el que la CPU necesita realizar varias operaciones de transmisión de datos.

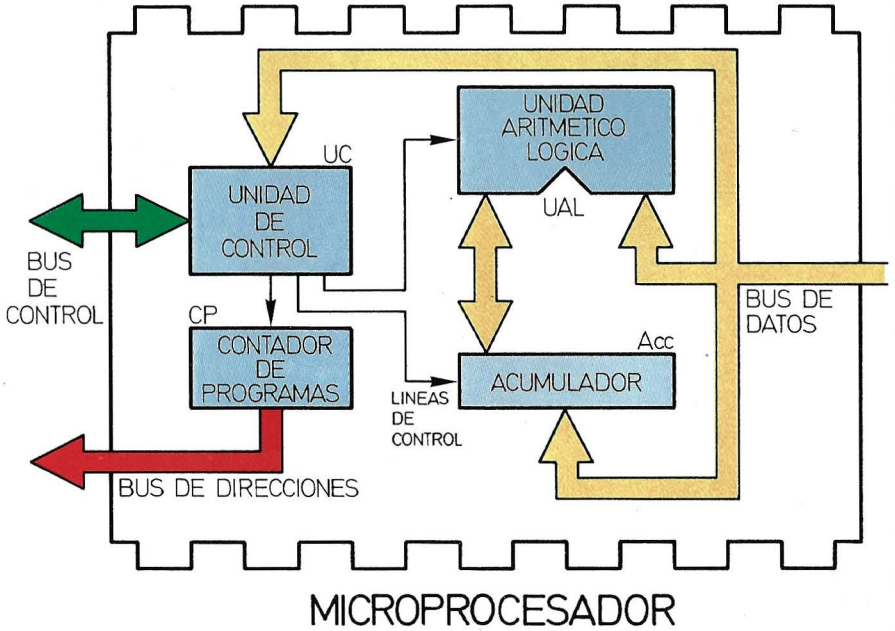
La segunda misión fundamental se concreta en el cálculo de nuevos datos a partir de los disponibles. Algo semejante a realizar la función de "calculadora" dentro del ordenador. Sin lugar a dudas, ambas actividades están mutuamente relacionadas: para sumar dos datos, por ejemplo, es necesario localizarlos en memoria, extraerlos de la misma, y hacerlos llegar al circuito sumador.

A menudo suelen separarse los circuitos de la CPU dedicados a estos dos cometidos. La zona que resuelve las operaciones con los datos recibe el nombre de *unidad aritmético-lógica* (UAL). Esta, por lo general, sólo es capaz de sumar, restar y realizar algunas operaciones lógicas (AND y OR). Las operaciones se efectúan sobre los datos contenidos en registros especiales y el resultado se guarda en un registro específico denominado *acumulador.*

La *unidad de control* (UC) es la otra zona básica de la CPU. Tal como su nombre indica, ésta es la que se encarga de controlar y supervisar la actividad global del ordenador. Se ocupa de hacer llegar los datos a los registros correspondientes, de enviar el resultado del acumulador a la zona de memoria adecuada y, en general, de trazar el ritmo de trabajo de la máquina.

La característica más importante de un ordenador radica en su aptitud para recibir una programación. Ello corre a cargo de una secuencia de órdenes —el programa— que definen con todo detalle las tareas que debe realizar. Las órdenes se codifican de forma inteligible para el aparato y se guardan en la memoria del mismo.

Para ejecutar un programa, la CPU ha de realizar múltiples operaciones elementales. En primer lugar, ha de "leer" la referida lista de instrucciones. Para ello empezará extrayendo de la memoria el primer dato coincidente con la primera instrucción codificada. La CPU dispone de capacidad suficiente para



Estructura interna elemental de un microprocesador.

por K=K+1. En tal caso, la ejecución conducirá al siguiente resultado:

```
RUN <CR>
334 KM A 8010 PTAS
■
```

EL PROBLEMA DE LOS BOTES DE CONSERVA

El segundo ejemplo se ocupará de acometer un cálculo de optimización. El problema consiste en diseñar un recipiente que, con el mismo contenido, utilice la mínima cantidad de material. Se partirá de un bote cilíndrico, cuyo volumen es de medio litro, al que se le variarán las dimensiones de altura y anchura.

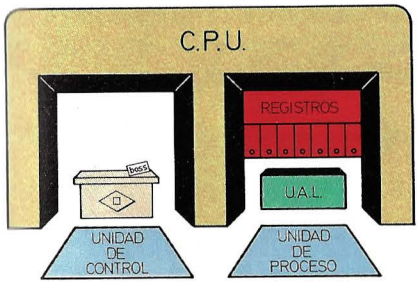
Para medir la cantidad de material a emplear, hay que centrar la atención en la superficie del bote. Esta puede referirse a dos magnitudes: altura y radio de la base. La superficie total será la suma de las dos bases más el área lateral. Cada base tiene un área de PI*R^2, siendo R el radio. A su vez, la superficie lateral será igual a la longitud de la circunferencia de la base multiplicada por la altura del bote. En definitiva, y dado que el perímetro es igual a 2*PI*R, la superficie lateral coincidirá con

interpretarla. Tal actividad se apoya en el denominado decodificador de instrucciones, que forma parte de la unidad de control. Una vez "asimilada" la orden, hay que planificar la secuencia de acciones elementales que darán pie a su ejecución.

Así, por ejemplo, cursar la orden BASIC LET A=B+C, exige a la unidad de control acometer los siguientes pasos:

1. Localizar la zona de memoria donde se encuentran los datos A y B.
2. Transmitir dichos valores, tras leerlos en la memoria, a los registros de la unidad aritmético-lógica (UAL).
3. Ordena a la UAL que opere la suma de ambos datos.
4. Recoger el resultado del acumulador, y llevarlo a la zona de memoria reservada a la variable A.

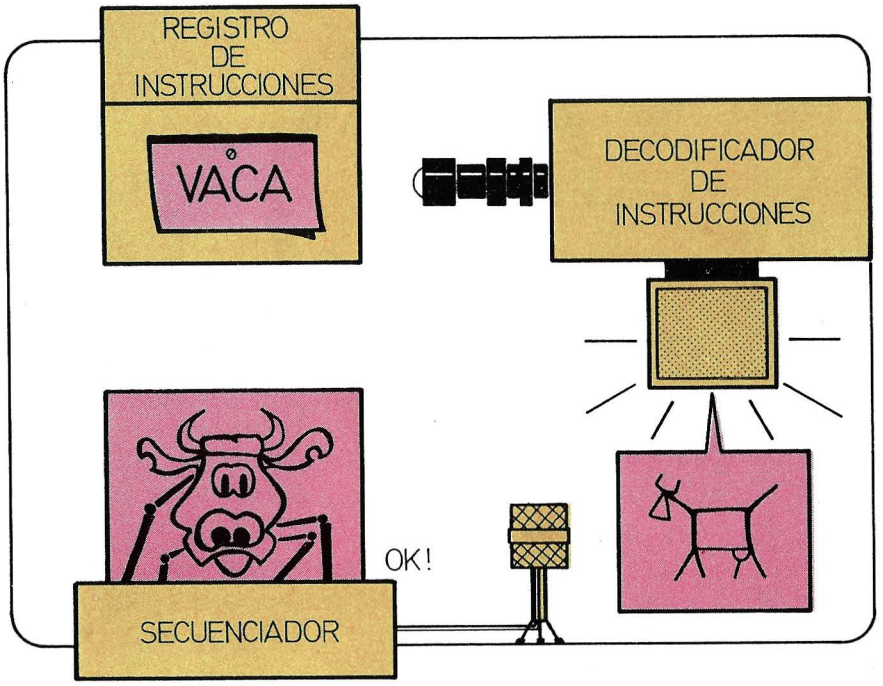
El orden de las sucesivas tareas



Dentro del microprocesador cabe diferenciar dos zonas básicas: la unidad de control y el bloque de proceso. Este último está constituido por la unidad aritmético-lógica y una serie de registros de trabajo.

elementales es estricto; no se puede realizar la suma sin antes disponer de los sumandos. Para ello, la CPU dispone de una especie de reloj interno que le permite encadenar los diversos pasos o microórdenes.

La unidad de control ha de supervisar



Los elementos básicos de la unidad de control son el registro de instrucciones (almacena la orden en curso), el decodificador de instrucciones (interpreta y decodifica el significado de la orden) y el secuenciador (encargado de activar las acciones necesarias para su ejecución).

todo el proceso y conocer en cada instante qué tareas se han realizado y cuáles quedan por cursar. Asimismo, debe saber cuál es la siguiente instrucción a leer y ejecutar.

Otro de los elementos primordiales de la CPU es el *contador de programas*. Se trata de un registro que indica en todo momento cuál es la dirección de memoria en la que está almacenada la próxima instrucción a ejecutar.

En síntesis, la unidad de control consta de los siguientes elementos básicos:

Registro de instrucciones, en el que se guarda la instrucción en curso. La CPU la traslada de la memoria a este registro para "tenerla más a mano". Aquí es donde puede proceder a su identificación.

Decodificador de instrucciones. Se trata de un circuito capaz de "reconocer" la instrucción en curso. Este inspecciona el contenido del registro de instrucciones y determina de qué orden se trata.

Secuenciador. Este circuito es el encargado de dar curso a la acción identificada por el decodificador. Una vez que éste ha interpretado la instrucción recibida, entra en actividad el secuenciador, quien se responsabiliza de ejecutarla. Para ello genera las microórdenes adecuadas (traspaso de datos, operación de la UAL...).

H*2*PI*R, llamando H a la altura del bote. Están ya definidos todos los ingredientes cuya suma determinará la superficie total del recipiente:

S=2*(PI*R↑2)+H*2*PI*R

Esta fórmula se puede reducir sacando el factor común a 2*PI*R, de donde resultará:

S=2*PI*R(R+H)

Lo único que hace falta es determinar los valores de R y H. Estos son los que han de variarse para encontrar la solución. Si bien, como quiera que el volumen del bote es fijo (medio litro), ambos estarán relacionados. La fórmula que relaciona el radio y la altura es precisamente la que proporciona el volumen:

V=PI*R↑2*H

Dado que el volumen es un dato fijo, lo que conviene es aislar la altura en función del radio (o viceversa). Tomando ya el valor de V igual a 500 cc. (medio litro) la expresión final será la que sigue:

H=500/(PI*R↑2)

De esta forma, variando únicamente el radio, se puede determinar la altura y la superficie del recipiente. Tras esta amplia disertación matemática, se llega finalmente al cuerpo del programa:

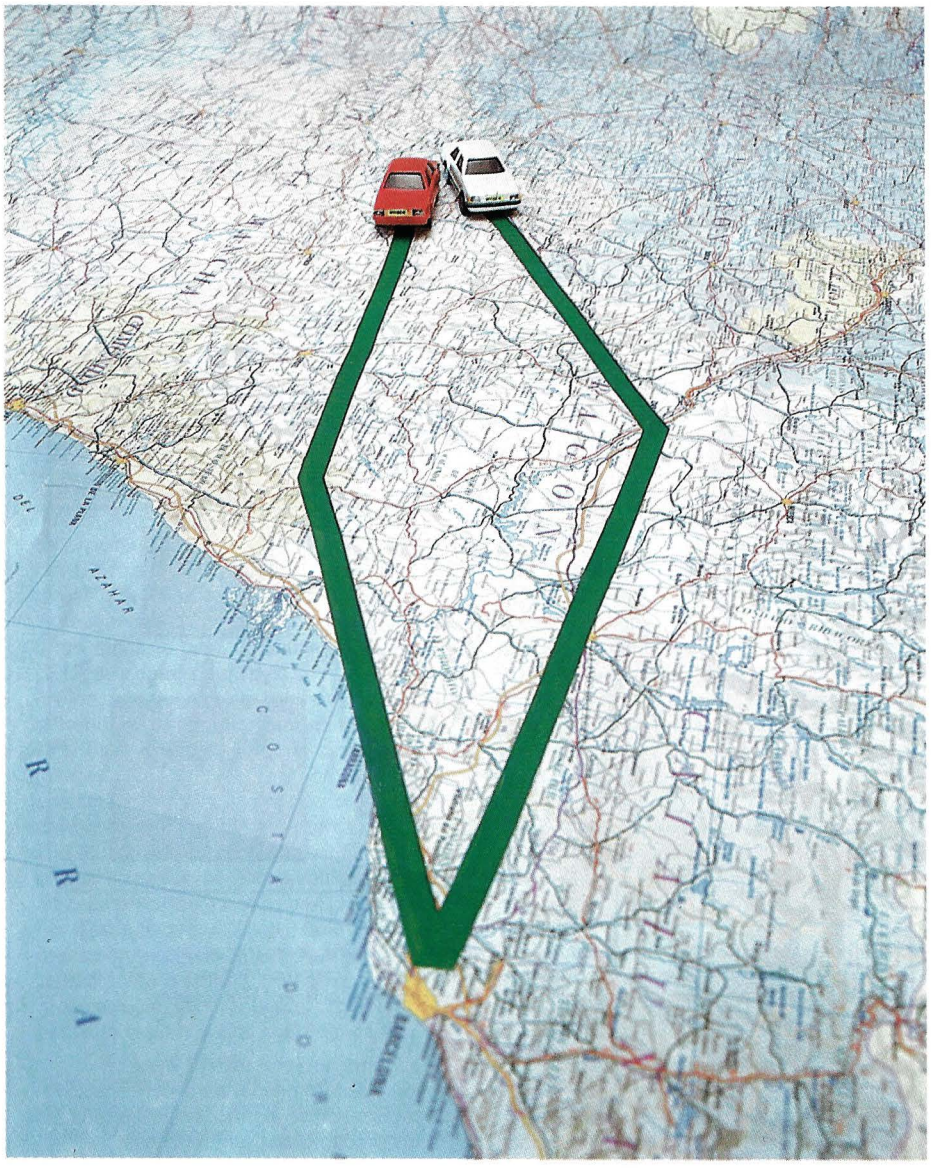
```
40 FOR R=1 TO 10
50 H=500/(PI*R↑2)
60 S=2*PI*R*(R+H)
80 NEXT R
■
```

Como se observa, el valor de R se hace variar entre uno y diez.
Hacen falta aún algunas líneas de programa: para mostrar los resultados, visualizar la cabecera de los mismos y para definir el valor del número PI:

```
20 PRINT "RADIO","ALTURA","SUPERFICIE"
30 PI=3.1416
40 FOR R=1 TO 10 STEP 0.5
50 H=500/(PI*R↑2)
60 S=2*PI*R*(R+H)
70 PRINT R,H,S
80 NEXT R
90 END
```

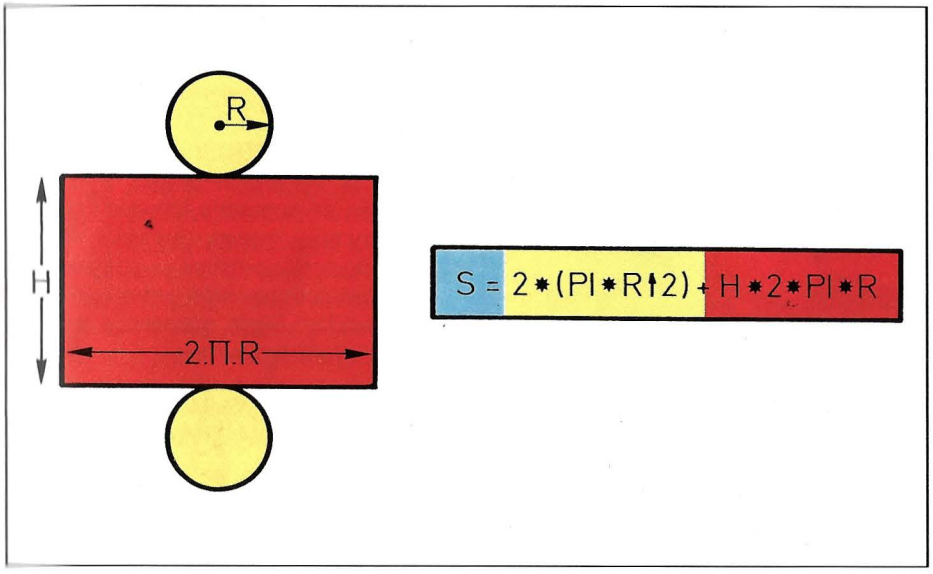
Una vez completado el programa puede ya introducirse la orden RUN para que sea ejecutado por el ordenador. Este será el resultado:

RADIO	ALTURA	SUPERFICIE
1	159.155	1006.28
1.5	70.7354	680.804
2	39.7886	525.133
2.5	25.4647	439.27
3	17.6838	389.882
3.5	12.9922	362.683
4	9.94716	350.531
4.5	7.85948	349.457
5	6.36618	357.08
5.5	5.26131	371.885
6	4.42096	392.862
6.5	3.76697	419.311
7	3.24805	450.734
7.5	2.82941	486.763
8	2.48679	527.125
8.5	2.20283	571.608
9	1.96487	620.05
9.5	1.76349	672.322
10	1.59155	728.32

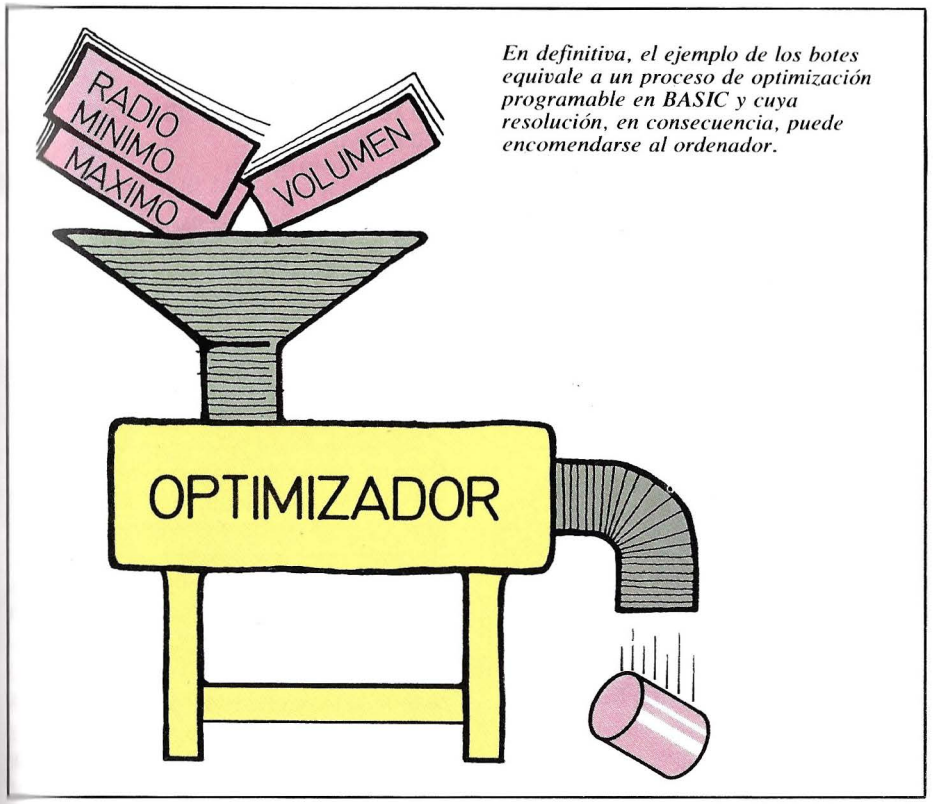


La eficaz herramienta que es el ordenador, debe ser capaz de construir una tabla comparativa del coste de cada fórmula de alquiler para distintos kilometrajes.

A la vista de la tabla, se observa de inmediato que la menor superficie se encuentra entre los valores del radio comprendidos entre 4 y 5 cm. Para hallar un valor más exacto construiremos un nuevo programa. En esencia, coincidirá con el anterior con la única salvedad que supone cambiar los valores límite y el incremento. El margen a inspeccionar es el situado entre los valores del radio delimitados por 4 y 5 cm. En consecuencia, éstos serán los límites del bucle FOR. El incremento se reducirá, asimismo, a 0.05 en la zona STEP de la instrucción FOR.



El área total del bote se calcula a partir de dos datos: radio y altura. La resolución del problema pasa por comparar los distintos valores de superficie hasta determinar cuál es la superficie mínima.



En definitiva, el ejemplo de los botes equivale a un proceso de optimización programable en BASIC y cuya resolución, en consecuencia, puede encomendarse al ordenador.

```
20 PRINT "RADIO","ALTURA","SUPERFICIE"
30 PI=3.1416
40 FOR R=4 TO 5 STEP 0.05
50 H=500/(PI*R ↑ 2)
60 S=2*PI*R*(R+H)
70 PRINT R,H,S
80 NEXT R
90 END
```

En este caso, la nueva tabla confeccionada por el programa será la siguiente. En ella se observa que el radio ideal es 4,3 cm. La altura correspondiente es de aproximadamente 8,6 cm. y la superficie igual a 348,734 centímetros cuadrados.

RADIO	ALTURA	SUPERFICIE
4	9.94716	350.531
4.05	9.70307	349.974
4.1	9.46785	349.523
4.15	9.24108	349.176
4.2	9.02237	348.931
4.25	8.81132	348.784
4.3	8.6076	348.734
4.35	8.41086	348.779
4.4	8.22079	348.915
4.45	8.03709	349.142
4.5	7.85948	349.457
4.55	7.68769	349.858
4.6	7.52147	350.344
4.65	7.36059	350.912
4.7	7.20482	351.562
4.75	7.05393	352.291
4.8	6.90774	353.098
4.85	6.76605	353.982
4.9	6.62867	354.941
4.95	6.49543	355.974
5	6.36617	357.08

Desde luego, el programa puede también encargarse de decidir cuál es la solución óptima. Para ello, debe encontrar el valor mínimo de la superficie. Dicho valor puede determinarlo comparando sucesivamente los contenidos de la variable S. En orden a facilitar su tarea, es preciso que entren en escena otras dos variables SMIN y RMIN. En la primera se guardará el valor mínimo de S resultante de las comparaciones parciales. A su vez, RMIN almacenará el radio correspondiente a dicha área mínima.
Ello se traduce en la incorporación de una nueva línea:

```
75 IF S < SMIN THEN SMIN=S:RMIN=R
```

Esta línea irá comparando los valores de S

"El problema de los botes de conserva"... un problema de optimización en el que el ordenador demuestra su amplia capacidad para los cálculos reiterativos.

e introduciendo el menor de ellos en SMIN, mientras que en RMIN quedará el radio correspondiente.
El único inconveniente estriba en la primera ejecución del bucle. SMIN ha de tener un valor inicial. Si dicho valor es inferior a todos los de S, nunca llegará a cumplirse la condición S<SMIN. Por lo tanto, hay que extremar el cuidado al elegir el valor inicial de SMIN. Para evitar cualquier problema, se inicializará a un valor suficientemente alto, por ejemplo:

```
35 SMIN=50000
```

Por supuesto, al concluir la ejecución hay que mostrar en la pantalla el valor del radio mínimo. De ello se ocuparán algunas instrucciones PRINT.

```
VOLUMEN?500
RADIO MINIMO?4
RADIO MAXIMO?5
RADIO        ALTURA      SUPERFICIE
4            9.94716     350.531
4.05         9.70307     349.974
4.1          9.46785     349.523
4.15         9.24108     349.176
4.2          9.02237     348.931
4.25         8.81132     348.784
4.3          8.6076      348.734
4.35         8.41086     348.779
4.4          8.22079     348.915
4.45         8.03709     349.142
4.5          7.85948     349.457
4.55         7.68769     349.858
4.6          7.52147     350.344
4.65         7.36059     350.912
4.7          7.20482     351.562
4.75         7.05393     352.291
4.8          6.90774     353.098
4.85         6.76605     353.982
4.9          6.62867     354.941
4.95         6.49543     355.974
5            6.36617     357.08

SOLUCION OPTIMA:
RADIO=4.3
ALTURA=8.6076
SUPERFICIE=348.734
```

```
 90 PRINT "SOLUCION OPTIMA:"
100 PRINT "RADIO=";RMIN
110 PRINT "ALTURA=";500/(PI*RMIN ↑ 2)
120 PRINT "SUPERFICIE=";SMIN
```

Aún cabe pulir en mayor grado el programa. Se puede contemplar la posibilidad de alterar el volumen de los botes. Para ello no tomaremos el dato constante 500; en su lugar, se introducirá la variable V. Esta se inicializará por medio de una instrucción INPUT.

```
10 INPUT "VOLUMEN";V
```

También es necesario introducir los valores mínimo y máximo del radio. En caso contrario podría omitirse el radio idóneo al no estar comprendido entre los límites establecidos. Serán necesarias dos nuevas líneas de programa:

```
14 INPUT "RADIO MINIMO";A
15 INPUT "RADIO MAXIMO";B
```

Una vez completo, éste adopta el siguiente aspecto:

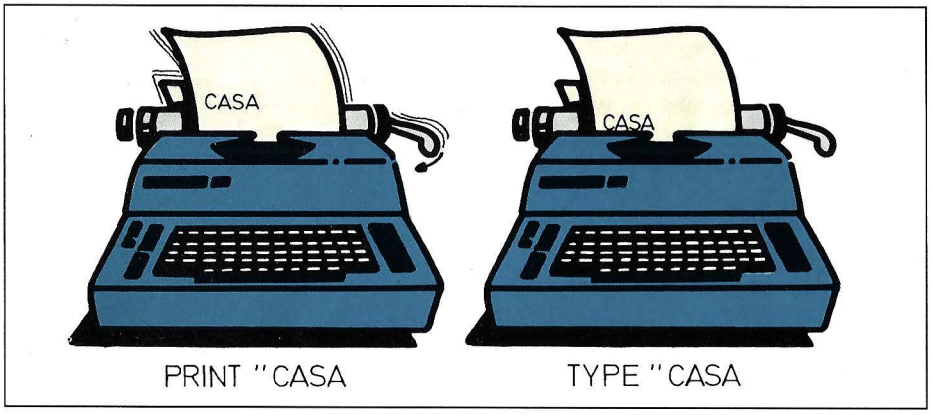
```
  1 REM *LOS BOTES*
 10 INPUT "VOLUMEN";V
 14 INPUT "RADIO MINIMO";A
 15 INPUT "RADIO MAXIMO";B
 20 PRINT "RADIO","ALTURA","SUPERFICIE"
 30 PI=3.1416
 35 SMIN=50000
 40 FOR R=A TO B STEP (B-A/20)
 50 H=V/(PI*R ↑ 2)
 60 S=2*PI*R*(R+H)
 70 PRINT R,H,S
 75 IF S < SMIN THEN SMIN=S:RMIN=R
 80 NEXT R
 90 PRINT "SOLUCION OPTIMA"
100 PRINT "RADIO=";RMIN
110 PRINT "ALTURA=";V/(PI*RMIN^2)
120 PRINT "SUPERFICIE=";SMIN
130 END
```

Cabe observar que tal como se ha señalado en un párrafo anterior, el valor 500 que aparecía en las líneas 50 y 110, se ha sustituido por la variable V. En la línea 40 se han tomado como límites del bucle los valores A y B, introducidos en las líneas 14 y 15. En ese mismo bucle, se ha fijado el incremento en un valor (B−A)/20. Ello hará que en la tabla aparezcan 20 valores igualmente espaciados.
La ejecución del programa conduce al resultado que aparece en el cuadro adjunto.

Logo (13)

La comunicación con el exterior

Hasta el momento, el diálogo con el ordenador en lenguaje LOGO se reducía a introducir las órdenes oportunas a través del teclado. El programador comunicaba, de esta forma, sus deseos a la máquina. El resultado podía verse reflejado en la pantalla de dos formas: por medio de trazos creados por la tortuga o en forma de palabras (o listas) escritas en la pantalla por medio de instrucciones PRINT.
En este capítulo se amplían los métodos de comunicación con el aparato; desde el control total de textos en pantalla hasta la posibilidad de introducir datos durante la ejecución de un procedimiento.



La diferencia entre los comandos PRINT y TYPE reside en que el primero provoca un salto a la siguiente línea tras escribir el texto indicado.

MANEJO DEL TEXTO

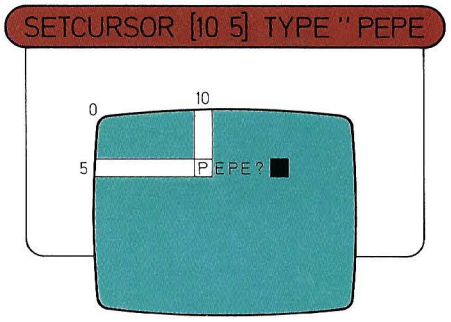
El comando que se ha utilizado hasta ahora para la visualización de textos es PRINT. Este admite un dato de entrada que puede ser una palabra, una lista o un dato numérico. Por ejemplo:

```
PRINT "CRISTINA
CRISTINA

PRINT 43
43

PRINT [1 2 3 4]
1 2 3 4
```

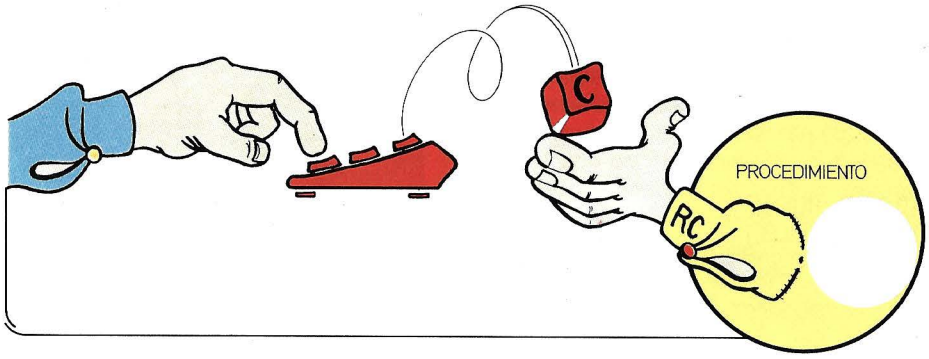
Cada vez que se ejecuta una orden PRINT, se salta a una nueva línea de la pantalla. El siguiente procedimiento ilustra dicha particularidad.

```
TO LEER
PRINT "JUAN
PRINT "LEE
PRINT "SOFTWARE
END
```

Al ejecutar este procedimiento se obtendrá la siguiente respuesta en la pantalla:



La orden SETCURSOR permite situar la visualización en cualquier punto de la pantalla, sin más que especificar las coordenadas en su argumento.



El operador RC se utiliza para recoger en un procedimiento el carácter introducido a través del teclado.

```
JUAN
LEE
SOFTWARE
```

Para visualizar todas las palabras en una misma línea, habría que agruparlas dentro de una lista; aunque ello no siempre es posible. Por lo demás, una vez escrita una parte del texto, no cabe la posibilidad de proseguir la escritura a continuación de la última palabra visualizada.

```
TO LEER :BUENO
PRINT [JUAN LEE]
IF :BUENO="SOFTWARE [PRINT :BUENO]
END
```

En el ejemplo, la palabra SOFTWARE quedará siempre una línea por debajo de las restantes.

```
LEER "SOFTWARE
JUAN LEE
SOFTWARE
```

Para evitar tal circunstancia, hay que recurrir a un comando que no ejecute un salto de línea después de realizar la escritura. Este comando es TYPE. El ejemplo precedente puede modificarse incluyendo el nuevo comando:

```
TO LEER :BUENO
TYPE [JUAN LEE]
IF :BUENO="SOFTWARE [PRINT :BUENO]
END
```

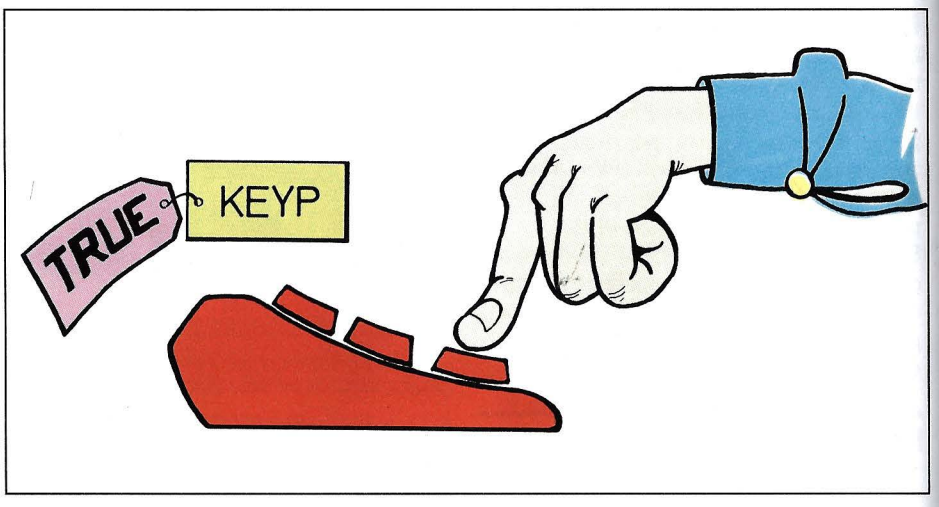
Ahora, su ejecución conducirá a la siguiente presentación en pantalla:

```
LEER "SOFTWARE
JUAN LEESOFTWARE
```

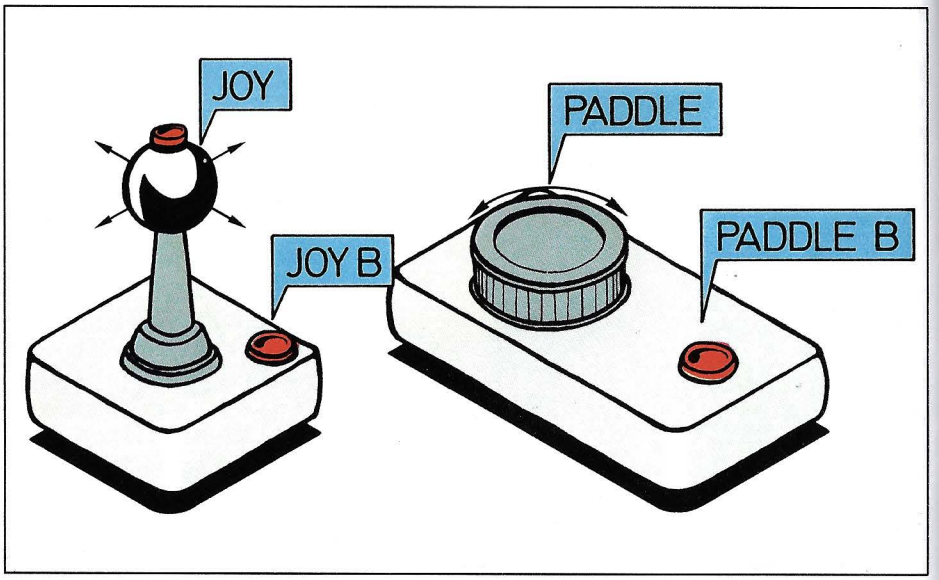
El resultado no es del todo satisfactorio, ya que sería aconsejable dejar un espacio en blanco entre las dos últimas palabras. Para visualizar dicho espacio en blanco, hay que utilizar el signo \ (backslash). Con esta nueva modificación, nuestro reiterado ejemplo quedará ya en perfecto orden:

```
TO LEER :BUENO
TYPE [JUAN LEE]
TYPE "\
IF :BUENO="SOFTWARE [PRINT :BUENO]
END
```

visualizando ya el mensaje en la forma deseada:



Recurriendo a la orden KEYP, es posible detectar cualquier acción que se produzca sobre el teclado. Dicho operador responde con TRUE (cierto) cuando se acciona una de las teclas.



El lenguaje LOGO dispone de funciones específicas para el control de periféricos especiales como, por ejemplo, joysticks y paddles.

LEER "SOTFWARE
JUAN LEE SOFTWARE

En definitiva, TYPE permite encadenar datos en la misma línea de presentación.

Tanto TYPE como PRINT admiten más de un dato de entrada, empleando el método de encerrar los datos y el comando entre paréntesis. Tal posibilidad es la que se ilustra a continuación:

(PRINT "JUAN"LEE"SOFTWARE)
JUAN
LEE
SOFTWARE

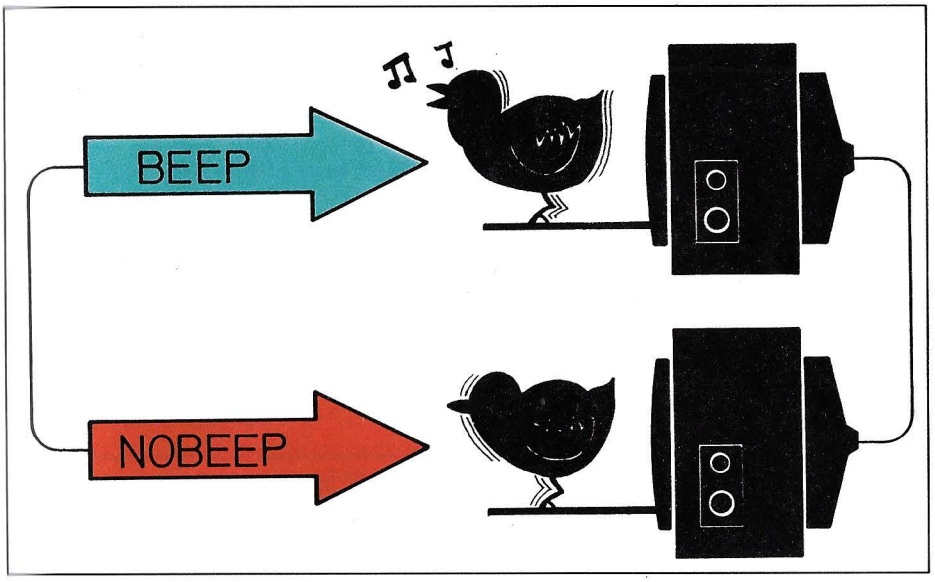
(TYPE "JUAN"LEE"SOFTWARE)
JUANLEESOFTWARE

Existe otro comando relacionado con la visualización de datos. Este es SHOW. Su función es idéntica a la de PRINT, con una única salvedad en la presentación de listas. Veamos un ejemplo.

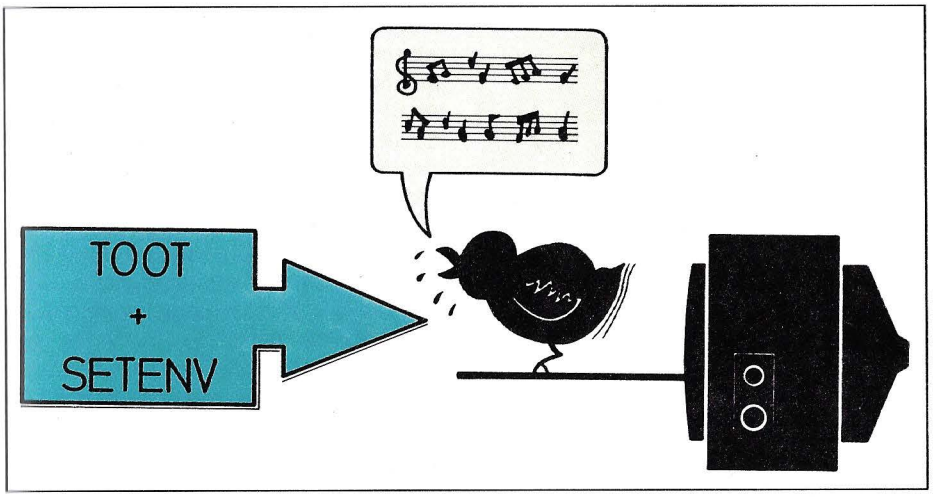
PRINT [DIA SEIS DE] PRINT "MAYO
DIA SEIS DE
MAYO

SHOW [DIA SEIS DE] SHOW "MAYO
[DIA SEIS DE]
MAYO

Por último, cabe mencionar al comando SETCURSOR, que permite elegir el punto de la pantalla en el que debe empezar la próxima impresión. El punto se especifica por medio de una lista de números: el primero representa la columna y el segundo la fila adecuadas. Los valores máximos de ambos parámetros dependerán de la resolución de la pantalla, en modo texto, característica de cada ordenador.



Los comandos de sonido más elementales y frecuentes en cualquier versión LOGO son BEEP y NOBEEP. Su única misión es la de conectar o desconectar el zumbador.



Algunas versiones del LOGO permiten generar sonidos complejos, por efecto de la acción conjunta de dos comandos: TOOT y SETENV.

RECOGIDA DE DATOS

Es posible introducir datos durante la ejecución de un procedimiento. Para ello es necesario instruir al ordenador oportunamente. La introducción manual de datos puede realizarse por medio del teclado o a través de un periférico especial: *joystick* (palanca de juegos) o *paddle* (potenciómetro de control).

En el primer caso se dispone de dos operadores adecuados para leer datos del teclado: RC y RL.RC (de Read Character, leer carácter) interrumpe la ejecución del procedimiento hasta que se pulsa una tecla. Una vez accionada, el carácter correspondiente a la tecla pulsada aparece como dato de salida de RC, continuando de inmediato la ejecución del procedimiento. Cabe señalar que dicho carácter no es visualizado en la pantalla.

La misión de RL es similar, si bien éste operador actúa con listas (Read List, leer lista). Con RL es necesario pulsar la tecla RETURN para indicar al ordenador que ha finalizado la introducción del dato. Los datos tecleados tras la ejecución de RL sí aparecen en pantalla.

Existe un tercer operador especializado en detectar cualquier acción sobre el teclado:

TABLA DE ORDENES-LOGO (I)

INSTRUCCION	COMETIDO	OPERADOR/ COMANDO
PRINT <dato>	Muestra en pantalla el dato y salta al principio de la línea siguiente	Comando
TYPE <dato>	Muestra en pantalla el dato sin saltar de línea	Comando
SHOW <dato>	Muestra en pantalla el dato. Si es una lista aparece entre corchetes	Comando
SETCURSOR <lista>	Sitúa el siguiente texto a escribir en la posición indicada por la lista de valores	Comando
RC	Devuelve el carácter pulsado en el teclado	Operador
RL	Devuelve la lista introducida por el teclado	Operador
KEYP	Responde con TRUE si en ese instante se pulsa una tecla	Operador

<dato>: Puede ser una palabra, lista o dato numérico. <lista>: Lista formada por dos números. Indican las coordenadas X e Y respectivamente.

TABLA DE ORDENES-LOGO (II)

INSTRUCCION	COMETIDO	OPERADOR/ COMANDO
JOY[<n>]	Devuelve un dato indicativo de la dirección en la que se desplaza la palanca del joystick	Operador
JOYB [<n>]	Devuelve TRUE si se acciona el botón de disparo del joystick	Operador
PADDLE [<n>]	Entrega un dato indicativo del giro dado en el botón de control del paddle	Operador
PADDLEB [<n>]	Devuelve TRUE si se pulsa el botón de disparo	Operador
BEEP	Conecta el zumbador	Comando
NOBEEP	Desconecta el zumbador	Comando

<n>: Número de dispositivo, en caso de admitir la conexión de más de uno.

KEYP (de key, tecla). Proporciona un dato de tipo lógico (TRUE) si, en efecto, se ha pulsado una tecla. A diferencia con los anteriores, KEYP no detiene la ejecución del procedimiento en el que se encuentra. El siguiente ejemplo hace uso de KEYP para detectar la pulsación de una tecla y de RC para identificar cuál es la tecla accionada. El objetivo es mover a la tortuga en las direcciones vertical y horizontal con las teclas W (arriba), Z (abajo), A (izquierda) y S (derecha).

```
TO MOVIL
IF KEYP [ANDA RC]
MOVIL
END

TO ANDA :HACIA
IF :HACIA="W [SETH 0]
IF :HACIA="S [SETH 90]
IF :HACIA="Z [SETH 180]
IF :HACIA="A [SETH 270]
FORDWARD 15
END
```

JOYSTICKS Y PADDLES

Por lo que respecta a la recogida de datos procedentes de periféricos especiales, la mayor importancia recae en las órdenes destinadas a joysticks y paddles, los dos periféricos especiales más comunes.
Los operadores implicados son JOY y JOYB para el joystick, y PADDLE y PADDLEB para el paddle.
JOY devuelve un número indicativo de la dirección en la que se ha desplazado la palanca del joystick. Estos valores no son comunes para todos los ordenadores, de ahí que, en cada caso, haya que consultar el manual propio de cada aparato. La misión de JOYB es detectar la pulsación del botón de disparo del joystick, entregando el dato TRUE si ocurre tal situación.
El operador PADDLE obtiene un dato numérico comprendido entre cero y un número positivo máximo. Este dato representa el ángulo girado en el botón de mando del paddle. La función PADDLEB es idéntica a la de JOYB, con la salvedad de que en este caso el botón es el del paddle.

SONIDOS

El uso de las capacidades sonoras del ordenador por medio de órdenes LOGO no está, en absoluto, estandarizado. Existen muy diversas alternativas que van desde la versión más simple, que sólo dispone de BEEP (inicia un zumbido) y NOBEEP (cancela el zumbido), hasta otras más complejas y con un mayor surtido de órdenes al efecto. En esta última categoría, suelen estar presentes los comandos TOOT y SETENV. El primero admite datos numéricos de entrada para establecer la frecuencia y el volumen del sonido. A su vez, SETENV controla la envolvente para obtener efectos sonoros más complejos. Si se dispone de equipos con más de un canal sonoro, se especificará también el canal deseado. Es obvio que dada la disparidad de dialectos LOGO en este punto, resulta imprescindible consultar el manual específico de cada equipo.

MS/DOS (1)

El más popular de los sistemas operativos para equipos de 16 bits

Son muy diversas las definiciones más o menos coloquiales y figurativas, aplicadas al sistema operativo: "la inteligencia elemental del ordenador" o "la mano invisible que guía el funcionamiento de la máquina" ... Semejantes calificativos se aplican de forma genérica al conjunto de los sistemas operativos. Una familia constituida por una ingente variedad de miembros, con mayor o menor presencia en el terreno informático.

¿Cuál es el motivo de la actual proliferación de distintos sistemas operativos destinados a microordenadores? La respuesta no es simple. En principio hay que considerar dos aspectos básicos que protagonizan tal dispersión. Por un lado la potencia y capacidad del microprocesador que rige el funcionamiento coordinado de cada ordenador; y por otro, la estrategia de la compañía fabricante del ordenador en lo relativo al tipo de aplicaciones y personas que van a utilizar sus productos.

Algunas consideraciones de tipo general contribuirán a vislumbrar los orígenes del sistema operativo que nos ocupa, el MS/DOS, y permitirán su comparación con otros sistemas operativos de su categoría.

En los primeros años de la década de los 70, cuando la microinformática se encontraba aún en sus albores, los equipos estaban desprovistos de sistema operativo. La introducción de datos e instrucciones se realizaba por medios rudimentarios (a veces, por medio de simples interrupciones), y bajo la total supervisión y control del propio usuario.

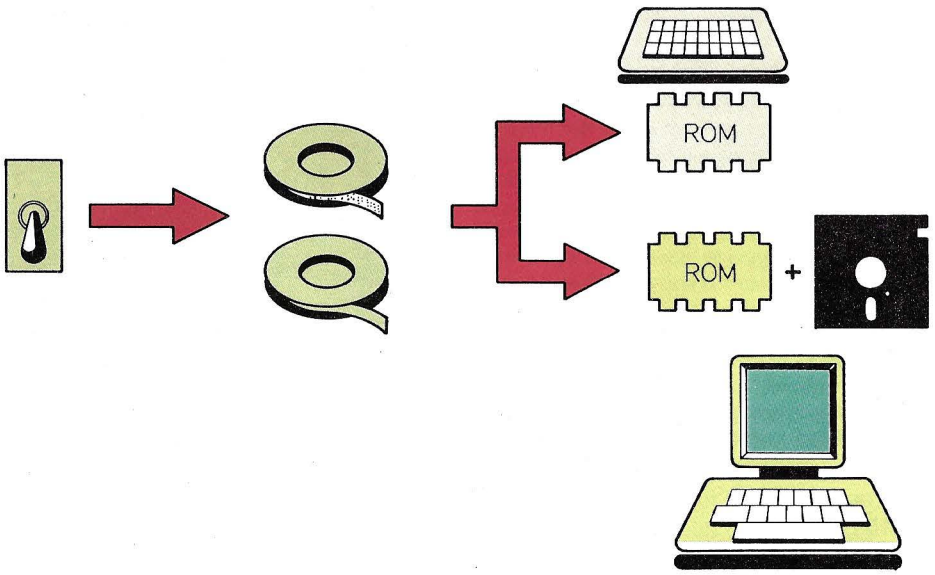
Este procedimiento, largo, tedioso y, sobre todo, propenso a la introducción de errores, era obvio que no podía prolongarse en el tiempo. Así, aparecieron rápidamente sistemas operativos de gran simplicidad, los cuales se almacenaban en memoria permanente (ROM) y que eran, en realidad, poco más que lo que hoy denominaríamos cargadores o "loaders". Estos programas permitían que el ordenador leyera de una cinta perforada —y, años más tarde de una cinta magnética—, un sistema operativo mucho más complejo. Esta idea, perfeccionada y refinada por una serie de consideraciones de tipo práctico, se ha adueñado del campo de los ordenadores personales, tanto para uso doméstico como profesional.

En el campo de los microordenadores para uso doméstico suele ser frecuente una recesión del sistema operativo a la memoria permanente de la máquina (ROM). Al ser normalmente escasa la cantidad de memoria disponible del equipo, no resulta muy atrayente la idea de ocupar gran parte de la misma con un sistema operativo potente y dejan tan sólo una mínima porción de memoria para los programas del usuario. Con tal limitación, el equipo perdería gran parte de su capacidad de trabajo. En este ámbito, cada fabricante suele crear su propio sistema operativo que, al ser desarrollado por el mismo, logra un más eficaz aprovechamiento de los recursos hardware del ordenador.

En el terreno de los ordenadores personales, de tipo profesional o de gestión, la idea primitiva de cargar en la máquina un sistema operativo más potente desde un medio de almacenamiento externo, es la que se ha consumado. Además, se manifiesta una acusada tendencia a que distintos equipos utilicen sistemas operativos comunes.

Los motivos que han empujado esta vía de evolución son, principalmente, tres. En primer lugar, la mayor capacidad de memoria de las nuevas máquinas permite destinar un cierto espacio al sistema operativo, sin por ello minimizar la memoria reservada al usuario.

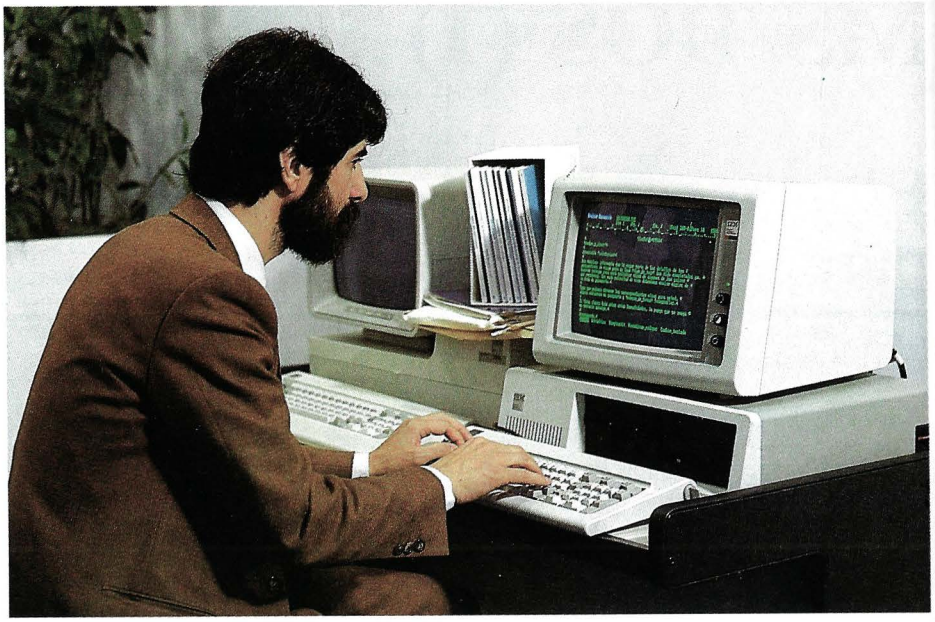
Por otra parte, para el fabricante del orde-



Con el nacimiento y progresiva evolución de los sistemas operativos, la introducción de los datos en el ordenador, así como el almacenamiento del propio sistema operativo, ha sufrido una total modificación.

nador, la creación de un sistema operativo competitivo y que haga pleno empleo de la capacidad de su equipo es una tarea larga y económicamente onerosa. A título ilustrativo, basta considerar que el S.O. empleado en el ordenador LISA, de la firma Apple Computers, necesitó un total de 200 años-programador para su desarrollo. Por lo demás, no hay que perder de vista la labor ingente necesaria para adaptar programas de aplicación estandarizados a un nuevo sistema operativo.

El tercer motivo, y quizás el más importante, reside en consideraciones de mercado. Hoy en día, con un mercado de microordenadores de tipo personal en constante crecimiento, y extendiéndose desde el ámbito doméstico hasta el empresarial, no parece muy consecuente la idea de que cada fabricante desarrolle su propio sistema operativo. La baza de las aplicaciones estandarizadas, creadas para su compatibilidad con los sistemas operativos de mayor resonancia queda fuera de semejante alternativa. Hay que tener en cuenta que muchos ordenadores no son expertos en informática y programación; su único deseo es que el ordenador les libere de tareas rutinarias a través normalmente de programas plenamente contrastados. En efecto, los paquetes de aplicación de mayor éxito comercial —por ejemplo, las hojas electrónicas o procesadores de texto— han sido desarrollados por compañías de software independientes; el hecho de que no puedan ser empleados en su propio ordenador opera en contra del fabricante.



El éxito del MS/DOS se debe, en gran medida, a su adopción como sistema operativo del ordenador personal de la IBM. En este equipo, las siglas MS/DOS se ven reemplazadas por las de PC/DOS.

LOS ORIGENES DEL MS/DOS

El sistema operativo MS/DOS es un producto creado por la firma americana Microsoft y destinado a ordenadores basados en los microprocesadores de 16 bits INTEL 8088 o INTEL 8086. Este sistema operativo se ha convertido en un verdadero estándar en el terreno de los equipos de 16 bits; al igual que el sistema operativo CP/M lo es para los ordenadores de 8 bits.

El éxito del MS/DOS tiene su raíz en el hecho de que la multinacional IBM lo eligiera como el sistema operativo de su ordenador personal: el IBM-PC. La gran aceptación del IBM-PC ha empujado al MS/DOS a la posición de liderazgo que hoy ocupa.

El camino que llevó a esta decisión es simple y directo, e ilustra las anteriores consideraciones. La elección por IBM del microprocesador INTEL 8088, deriva, en parte, de la peculiar arquitectura de este microprocesador. Aunque éste opera internamente con palabras de 16 bits, la entrada y salida de datos se realiza en bloques de tamaño byte (8 bits). Las ventajas que de ello derivan son claras. Por un lado, la posibilidad de utilizar circuitos simples y baratos, adecuados para trabajar asociados a un bus de 8 bits. Además del hecho de que el repertorio de instrucciones en lenguaje ensamblador para el procesador 8088, en un conjunto ampliado de las utilizadas por el microprocesador de

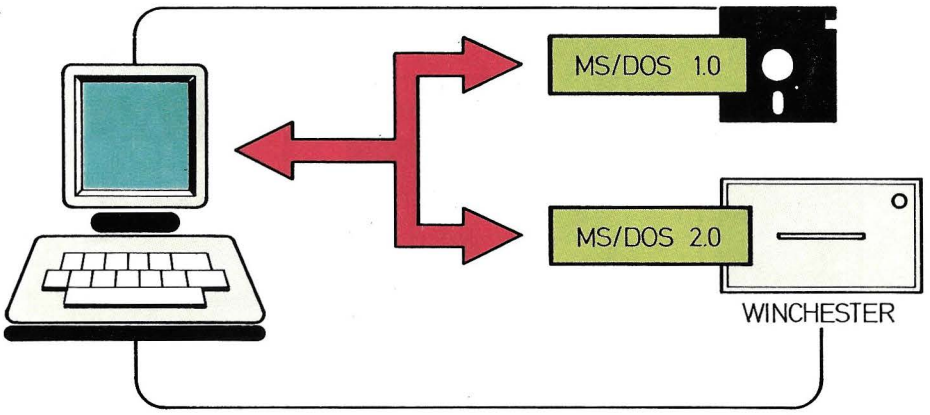
Al igual que el CP/M es el sistema operativo líder en el ámbito de los equipos de 8 bits, el MS/DOS es el sistema operativo imperante en el terreno de los equipos basados en un microprocesador de 16 bits. A la sombra del IBM-PC, han florecido un elevado número de ordenadores compatibles que, día a día, consolidan al MS/DOS en su liderazgo.

8 bits 8080. Ello significa que un gran número de programas desarrollados para el 8080, sólo necesitan ser reensamblados para su traslado al microprocesador de 16 bits 8088. De esta forma se accede, sin excesivas complicaciones, a un amplio abanico de programas de éxito comercial. Dado que el MS/DOS fue creado, precisamente, para gobernar al microprocesador 8088, la elección ofrecía pocas dudas.

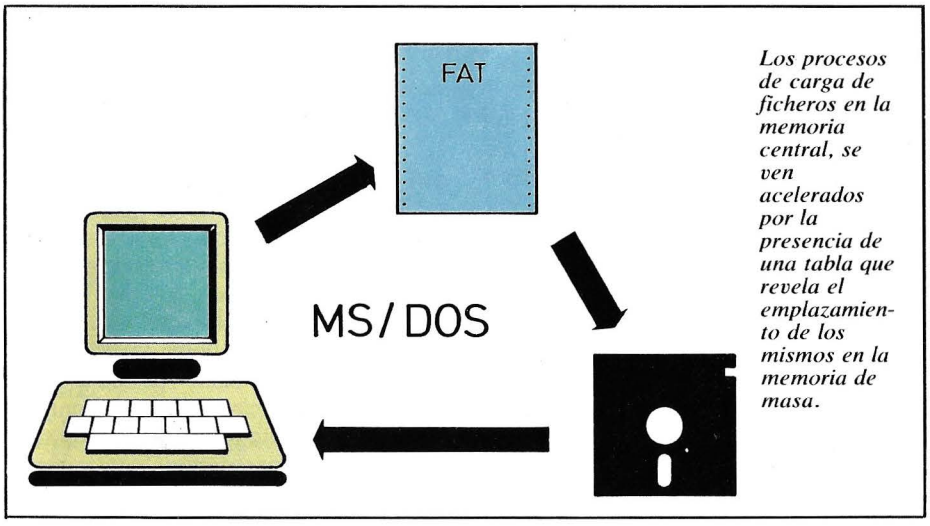
Conceptualmente, el MS/DOS guarda una gran similitud con el sistema operativo CP/M, ya que ha seguido sus pautas.

El MS/DOS está orientado esencialmente a la gestión de archivos en disco, de ahí su denominación: DOS (Disk Operating System). En su versión, DOS1.0, está configurado para operar con dos unidades de disco flexible, mientras que en la versión DOS2.0 se emplea para el control de unidades de disco rígido de tecnología Winchester.

Una gran ventaja de este aspecto del MS/DOS, es que cada uno de los distintos periféricos conectados al ordenador está considerado en un módulo propio del sistema operativo. Con ello, es obvio que el equipo puede adaptarse al uso de muy diversas categorías de periféricos. Este método presenta sin embargo una desventaja clara: está pensado más en función de las necesidades del ordenador que de las del propio usuario, lo que obliga a éste a conocer los diferentes comandos del sistema operativo, su función y su formato exacto.



Las sucesivas versiones del sistema operativo MS/DOS están orientadas al manejo de diversas unidades para el almacenamiento externo de información. La versión DOS 2.0 completa su capacidad de trabajo sobre discos flexibles, con la facultad de controlar unidades de disco rígido.



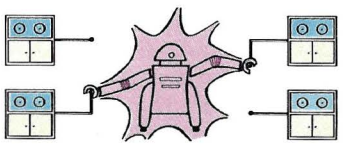
Los procesos de carga de ficheros en la memoria central, se ven acelerados por la presencia de una tabla que revela el emplazamiento de los mismos en la memoria de masa.

La comunicación entre ordenadores distantes

En la actualidad, la tendencia primaria en el campo de la informática es la de brindar a cualquier usuario la posibilidad de acceder a los recursos de un ordenador, aunque éste no se encuentre físicamente próximo. De ahí nace el denominado proceso distribuido, por el cual, máquinas de distinta capacidad y potencia se interrelacionan entre sí, compartiendo datos, programas y recursos. Dependiendo de la distancia que medie entre los ordenadores asociados, se emplean diversos métodos de comunicación, tanto a nivel lógico como físico. Entre los métodos adecuados para la comunicación a larga distancia, los dos más comunes son los que se conocen bajo apelativos de "conexión por paquetes" y "conexión por líneas".

En el primer caso, la información a transmitir se divide en partes, denominadas paquetes, cada uno de los cuales lleva una etiqueta en la que se indica su destino. En la línea de comunicaciones existen ciertos elementos, denominados controladores,

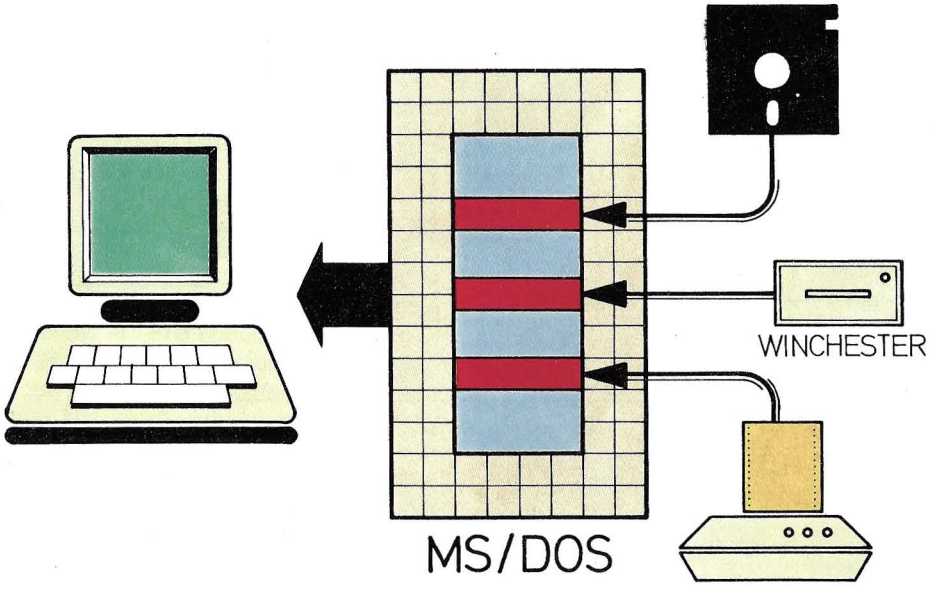
CONEXION POR PAQUETES



CONEXION POR LINEA

cuya función es leer las etiquetas y pasar el paquete a su destinatario, o al siguiente controlador más próximo. Cuando el paquete llega a su destino se decodifica y la información se almacena en el sistema.

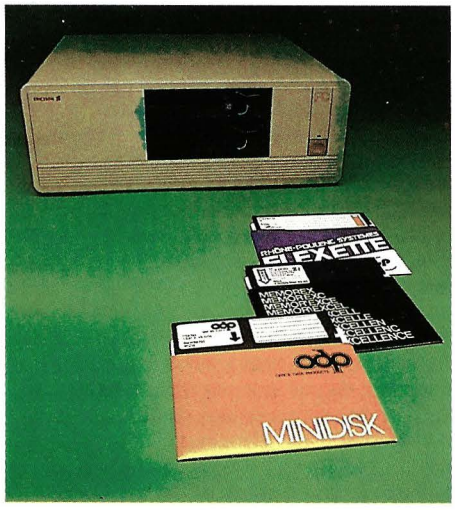
En el segundo método, el controlador se encarga de facilitar una comunicación directa entre los dos ordenadores que van a intercambiar información, durante un período de tiempo extremadamente breve. En el transcurso de este intervalo de tiempo, se envía toda la información a transmitir de una sola vez. Este método tiene la desventaja de que depende drásticamente del número de líneas disponibles para habilitar el diálogo. De ahí que pueda darse el caso de que se intente la comunicación y no haya ninguna línea disponible para establecerla en ese momento.



El sistema operativo MS/DOS considera por separado a los diversos tipos de periféricos conectables. Cada uno de ellos es controlado por un módulo distinto del sistema operativo.

COMPARACION ENTRE EL MS/DOS Y OTROS SISTEMAS OPERATIVOS

Desde luego, la orientación del MS/DOS hacia los equipos de 16 bits, marca una distinción activa respecto a los sistemas operativos destinados a los microprocesadores de 8 bits. Sin embargo, el MS/DOS también presenta notables diferencias con otros sistemas operativos destinados a microprocesadores de 16 bits; tal es el caso del UNIX, un producto de los laboratorios Bell. También son considerables sus diferencias con los sistemas operativos integrados —"user-friendly"— que equipan a los ordenadores Lisa y MacIntosh de la firma Apple. Veamos algunas diferencias esenciales que presenta el MS/DOS respecto a otros sistemas operativos de gran aceptación.



Tal y como revelan las siglas DOS (Disk Operating System), el MS/DOS está especialmente orientado a la gestión de archivos en disco.

• CP/M

Ambos sistemas operativos presentan una gran similitud a nivel general, aunque difieren en aspectos muy importantes. Básicamente, el CP/M se ve frenado por el reducido volumen de memoria que pone en sus manos un microprocesador de 8 bits. Ello se traduce en una disminución en el número de comandos a disposición del operador. La consecuencia práctica es que para llevar a cabo todas las funciones necesarias, es preciso completar cada comando con una serie de parámetros adicionales, con el consiguiente problema de memorización por parte del usuario. Asimismo, los requerimientos de memoria hacen que los mensajes de error sean crípticos y no muy numerosos.

Por contra, el MS/DOS ofrece un mayor número de comandos y con un formato más lógico, así como un repertorio de elaborados mensajes de error y subrutinas de comprobación. Una diferencia adicional es que este sistema operativo mantiene continuamente en memoria una tabla de localización de archivos (FAT). De esta forma, los accesos a los ficheros en disco son mucho más rápidos, al saber el sistema operativo donde tiene que posicionarse en el disco para localizar la información.

• UNIX

Este sistema operativo presenta la gran diferencia de estar escrito en un lenguaje de alto nivel, con lo cual es totalmente independiente del hardware. En consecuencia puede aplicarse a casi cualquier ordenador, desde personales hasta grandes equipos, con las consiguientes ventajas en la compatibilidad de programas. Otra diferencia sustancial reside en el hecho de que la interacción del usuario con el sistema operativo es posible canalizarla, de acuerdo con sus preferencias y necesidades, a través de menús. Ello simplifica en gran medida la tarea de dar órdenes a la máquina. El UNIX se desarrolló inicialmente para mini-ordenadores; su descenso al terreno de los microordenadores está condicionado por la característica apuntada al principio. En este ámbito, la eficacia del UNIX dependerá de la presencia en el equipo de un potente microprocesador que no ralentice los ciclos de respuesta; en este punto, el UNIX es un voraz consumidor de ciclos de máquina.

• S.Os. integrados para Apple Lisa y MacIntosh

Ambos ordenadores comparten un sistema operativo análogo. Este es único en su categoría. Se trata de un sistema operativo totalmente orientado al usuario y en el que el diálogo se reduce al desplazamiento de un indicador sobre la pantalla, gobernado por el ya popular "ratón". Las operaciones y procesos se ordenan movilizando el indicador sobre un grupo de menús e iconos (gráficos representativos de ficheros, procesos y acciones) presentes en la pantalla.

En definitiva es la propia filosofía de estos sistemas operativos la que se distancia por completo de la propia del MS/DOS y de cualquier otro S.O. orientado a la máquina.

Por lo demás, hay que señalar que los referidos sistemas operativos para los modelos Apple Lisa y MacIntosh, operan sobre un microprocesador con arquitectura interna de 32 bits, aunque con estructura de comunicación externa de 16 bits: el Motorola 68000.

PFS File-Report (1)

Un paquete estandarizado para la gestión de ficheros

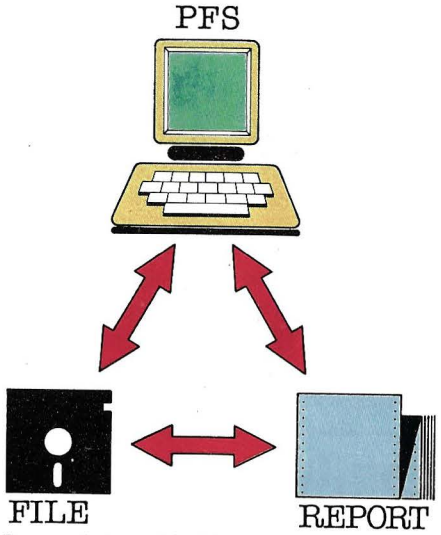
Al introducir las aplicaciones para la gestión de bases de dátos, se apuntaba la existencia de muchos productos para microordenadores que, aún sin llegar a constituir verdaderas bases de datos, realizaban algunas de sus funciones típicas. En general, a este grupo de aplicaciones cabe otorgarle la denominación de *gestores de ficheros*. En este capítulo da comienzo el estudio del paquete PFS File and Report; una aplicación encuadrable en esta categoría de productos software: sin ser una base de datos propiamente dicha, permite gestionar ficheros de forma eficiente y obtener informes a partir de su contenido. Dentro de sus múltiples versiones, destaca la adaptada al APPLE MacIntosh; ésta es precisamente la versión considerada en nuestro análisis práctico.

ASPECTOS BASICOS DEL PFS

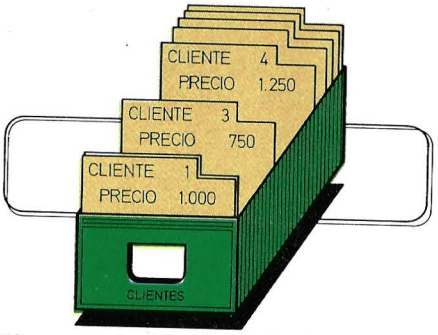
Como revelan los "apellidos" de este producto (File and Report), el PFS consta de dos zonas o aspectos fundamentales: gestión de ficheros y producción de informes. La independencia entre ambos subsistemas es tan elevada que pueden llegar a adquirirse y utilizarse por separado. No obstante, la mejor forma de aumentar el rendimiento de la información almacenada, es coordinando la actuación de los paquetes PFS-File y PFS-Report.

- *PFS-File*

Se trata de un programa de aplicación destinado a almacenar datos en ficheros y recuperarlos de forma cómoda y organizada. En síntesis, el PFS-File permite realizar tres procesos genéricos. El primero consiste en el diseño de la estructura con la que se guardarán los datos para cada unidad elemental de información (entendiendo por unidad elemental al ente sobre el que se informa, ya sea una persona, cliente, pieza, etc.). El segundo proceso se concreta en la propia introducción de los datos, de acuerdo al diseño anterior. Por último, el tercer proceso es el especializado en la recuperación de datos en cualquier orden (no necesariamente en el mismo orden en el que se introdujeron). Por supuesto, el diseño de la estructura con la que deben almacenarse los datos ocupa el primer escalón en el orden de las tareas a realizar. Sin embargo, la introducción y recuperación de datos se puede simultanear en cualquier orden.



Dentro de la aplicación PFS cabe considerar la presencia de dos paquetes complementarios: PFS-File y PFS-Report. El primero de ellos puede operar de forma independiente, dado que también incorpora funciones adecuadas para la edición de informes. No obstante, la mayor eficacia en la gestión de ficheros se obtiene con la asociación de ambos paquetes.



El proceso de trabajo con el PFS-File tiene un total paralelismo con los métodos para el tratamiento de ficheros tradicionales. En primer lugar se diseña la distribución de campos en cada ficha y, tras ello, se irán creando sucesivas fichas que pasarán a formar parte del fichero en cuestión. Por supuesto, las fichas pueden ser recuperadas y su contenido modificado a voluntad del usuario.

Para ello, el hardware exigible para el desarrollo de la aplicación, se concretan en un ordenador personal, la unidad o unidades de almacenamiento y el terminal de comunicación (pantalla y teclado).

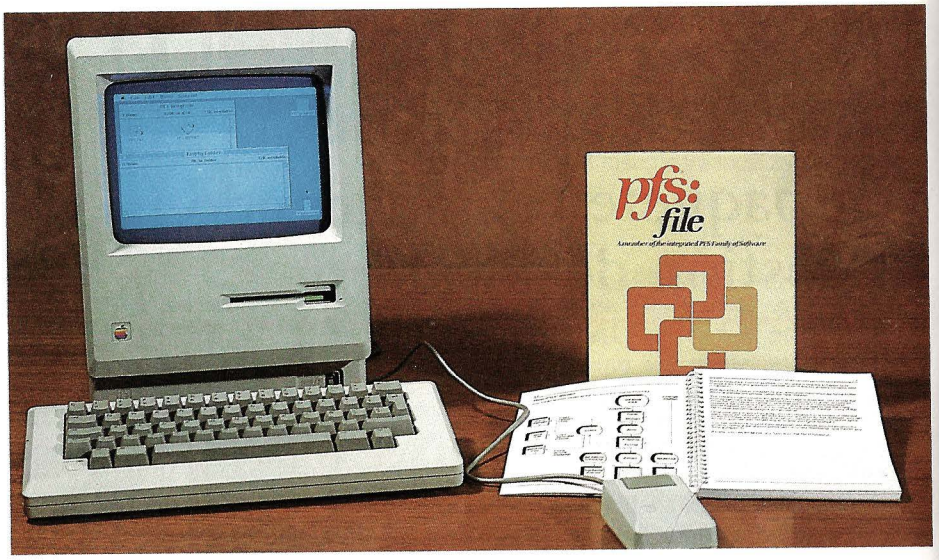
- *PFS-Report*

Al igual que en el caso anterior, el PFS-Report es un programa de aplicación que permite recuperar información de un fichero. La diferencia estriba en que el PFS-File permitía, fundamentalmente, la introducción de datos; a su vez, la recuperación debía realizarse siempre de acuerdo al diseño de la estructura de almacenamiento. En cambio, el PFS-Report es capaz de generar informes elaborados que, basándose en la información del fichero, contienen listados parciales o totales con una parte de datos originales y otros calculados a partir de éstos. Además, el formato del informe no tiene por qué coincidir con el diseño del fichero.

Los informes generados mediante el PFS-Report pueden llegar a tener múltiples columnas y un número de líneas indeterminado; éstos se pueden estructurar de tal

forma que se dividan en páginas homogéneas. El orden en el que se situarán las líneas, se determinará mediante una clasificación alfabética o numérica aplicada a cualquiera de los campos de datos. La única restricción que se impone a los ficheros para ser utilizados en un informe es que sean de tipo PFS.
Al igual que en el caso del PFS-File, son importantes las unidades de almacenamiento y las unidades de comunicación para el buen funcionamiento del PFS-Report. Si bien, en este caso, y debido a la naturaleza del resultado, cobra también importancia la impresora.

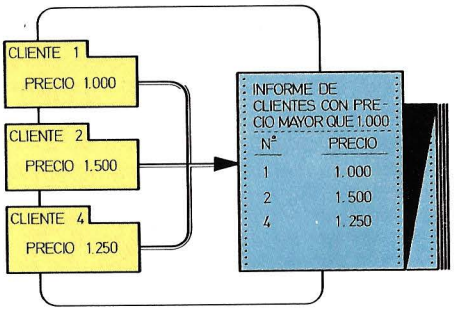
FUNCIONAMIENTO DEL PFS-FILE



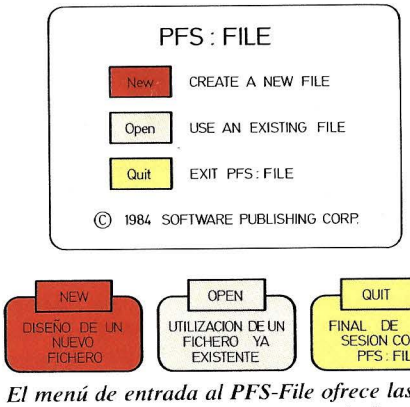
El PFS-File es un paquete de aplicación, estandarizado, adecuado para gestionar ficheros de forma eficiente y obtener informes selectivos de su contenido.

Después de introducir el disco con el programa PFS-File en la unidad correspondiente, y tras cargarlo en memoria, aparecerá un menú en la pantalla que ofrece al usuario tres opciones:

1. NEW (Create a new file: *crear un fichero nuevo)*
En este caso, será necesario retirar el disquete del programa, reemplazándolo por un disco de datos previamente inicializado. A continuación, se tecleará el nombre del fichero que se desee crear. El nombre puede estar integrado por cualquier cadena de hasta 40 caracteres, entre los que no puede incluirse el signo dos puntos (":"); tampoco está permitido que el nombre empiece por punto (".."). Para finalizar la asignación de nombre al fichero, hay que elegir dos opciones: OK que servirá para confirmar los pasos anteriores y CANCEL para anularlos.
Acto seguido es preciso diseñar el contenido de cada elemento del fichero. Al efecto aparecerá en la pantalla una página en blanco, encabezada por el mensaje "FILE IN PAGE 1 OF DESIGN" (se pueden utilizar hasta 32 páginas) y por el nombre que previamente se le haya otorgado al fichero. En dicha página se debe introducir el formulario que servirá para la captación o entrada de datos. El formulario será semejante al que podría definirse sobre una ficha tradicional de cartulina. En el diseño pueden emplearse tres tipos de formatos:
— Líneas de comentarios que facilitarán tanto la introducción como la obtención de información.
— Nombres para la identificación de cada uno de los campos de datos que constituirán la información almacenada en cada ficha o elemento.
— Espacios en blanco reservados para la introducción de los datos. Su presencia condiciona el número máximo de caracteres o dígitos de que constarán los distintos datos.
Estos tres tipos de líneas se deben cumplimentar, según el diseño que desee crearse, esgrimiendo las teclas para el desplazamiento del cursor y las teclas alfanuméricas apropiadas. Después de cada nombre identificador, hay que colocar el carácter (":") dos puntos.



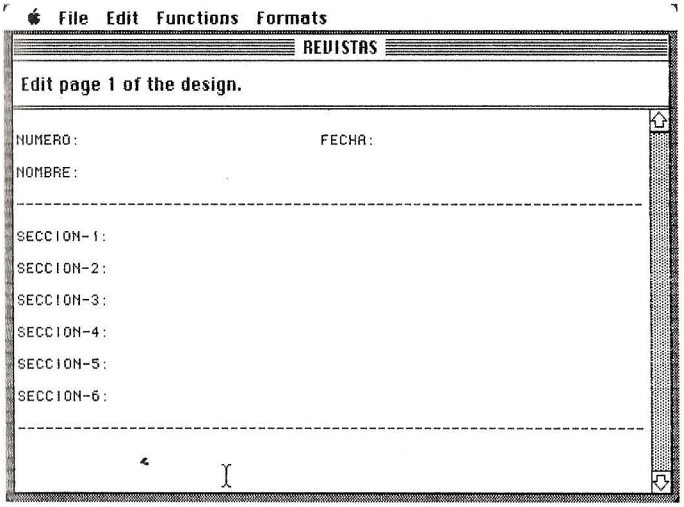
Continuando con la analogía adoptada en la figura precedente, cabe señalar que el objetivo del PFS-Report no es otro que la generación de informes escritos, partiendo del contenido de todas o de algunas de las fichas almacenadas.



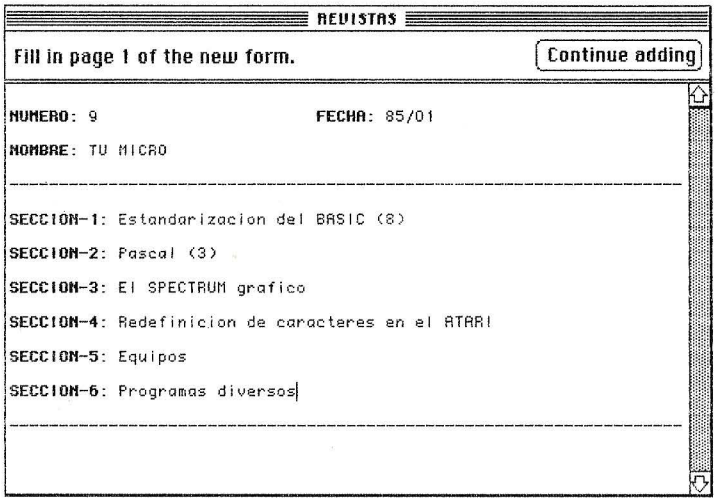
El menú de entrada al PFS-File ofrece las tres opciones genéricas que refleja la figura.

2. OPEN (Use an existing file: *utilizar un fichero ya existente)*
En este caso se omitirán las fases de "bautizo" y diseño del fichero, puesto que éste habrá sido definido previamente. Si se desea abrir directamente un fichero cuyo nombre es conocido, bastará con invocarlo. En cambio, si no se conoce su denominación, puede ordenarse la presentación en pantalla de los nombres de todos los ficheros existentes en el disco de datos utilizado. Dado que el disco puede contener un número elevado de ficheros y en la pantalla sólo pueden aparecer seis simultáneamente, puede ordenarse la rotación o "scrolling" de los mismos; esto es: al aparecer el nombre del séptimo fichero desaparecerá el del primero, al aparecer el del octavo desaparecerá el segundo, etc.

3. QUIT (Exit PFŞ: *salir del PFS-File)*
La finalidad de esta opción es trivial: sencillamente, provoca el abandono del programa, retornando el control al sistema operativo.



Diseño de la ficha correspondiente al ejemplo descrito en el texto. Esta contiene el formato que se utilizará para almacenar los datos relativos a las revistas coleccionadas por el usuario. (La figura es una reproducción directa de la pantalla del ordenador Apple Macintosh.)



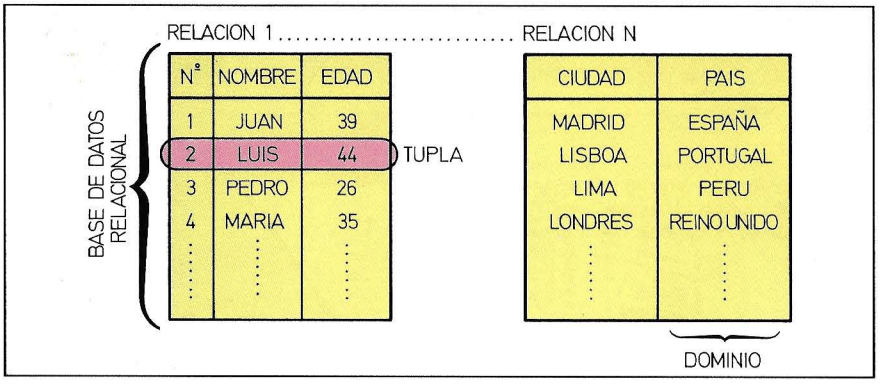
Una vez introducidos los datos en los distintos campos, el aspecto de la ficha será el que muestra la figura. (Reproducción directa de la pantalla del ordenador Apple Macintosh.)

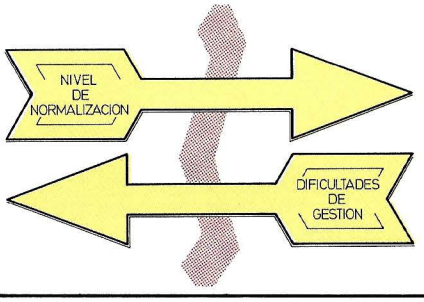
INCLUSION Y EXTRACCION DE DATOS DE UN FICHERO

Para concluir este primer capítulo dedicado al estudio del PFS-File, resulta adecuado ilustrar, mediante un sencillo ejemplo, el método a seguir para introducir y extraer datos por medio de la aplicación. Suponga que se ha diseñado un formulario para almacenar ciertos datos relativos a las revistas que el usuario colecciona. El formulario incluye campos con el número de orden de la revista entre las coleccionadas, su fecha de publicación, el nombre de la revista y el contenido de hasta seis posibles secciones de cada revista.
Para introducir la información de una de-

Modelo relacional de base de datos

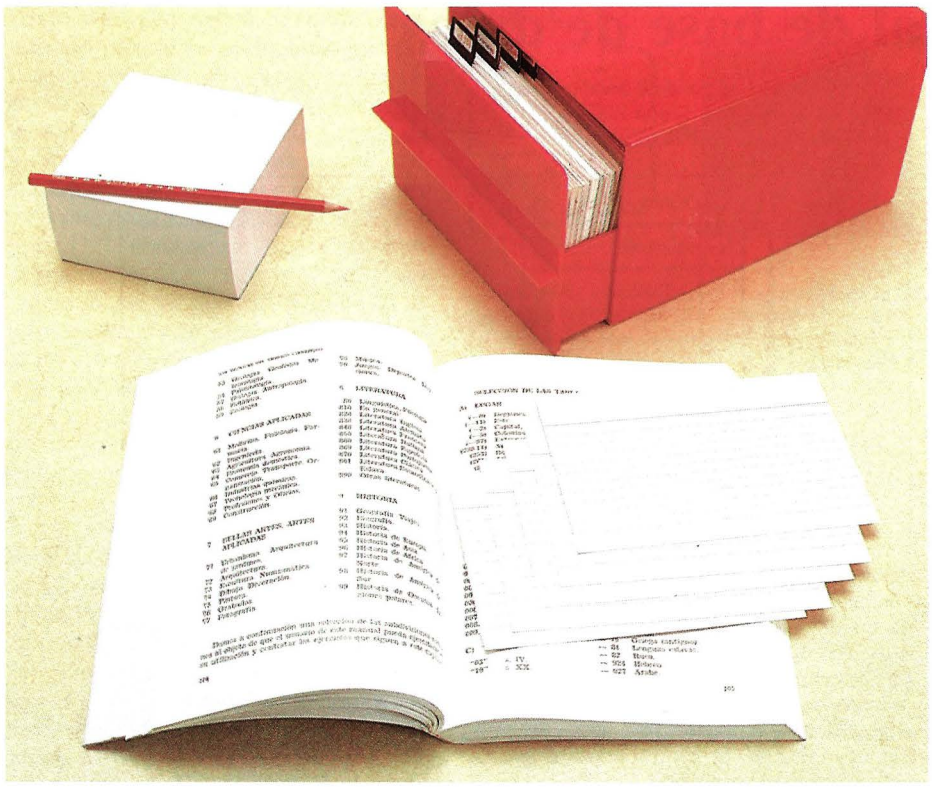
La relación es el elemento fundamental y definitorio de este modelo de base de datos. Intuitivamente, una relación puede asociarse a una matriz en la que cada columna tiene, además de un significado específico, un cierto dominio dentro del que puede tomar valores numéricos o alfanuméricos. Dentro de la matriz, cada fila (denominada habitualmente tupla) representa a un elemento de la relación.
Partiendo de este fundamento, puede definirse a la base de datos relacional como "un conjunto finito de relaciones variables con el tiempo, definidas sobre un conjunto finito de dominios".
El único problema que puede plantear este modelo estriba en la consecución de un buen diseño de relaciones. Ello supone asociar los datos en tratamiento a relaciones que garanticen una cómoda y eficaz actualización y obtención de los datos.
Para llegar a un buen diseño, puede adoptarse la teoría estudiada y formalizada por Codd en 1970. En síntesis, dicha teoría transforma una base de datos en la que las relaciones no deben satisfacer ninguna propiedad especial, en otra base de datos normalizada. Para operar esta transformación se van imponiendo sucesivamente una serie de condiciones. Con ello, se pasa del diseño inicial a sucesivos diseños que responden a la denominación de "primera forma", "segunda forma" y "tercera forma normal de Codd".
A medida que va aumentando el ordinal de normalización, la organización de las relaciones que forman la base resulta más adecuada para su explotación.
Sin lugar a dudas, el modelo relacional es uno de los más sencillos de llevar al terreno práctico. El único factor que obstaculiza su imposición frente a otros modelos menos eficientes, reside en que exige el empleo de potentes equipos informáticos.





BUSQUEDAS DE INFORMACION MEDIANTE EL PFS-FILE

Caracteres tecleados	Búsqueda
abc	Sólo los datos que coincidan con abc
. . abc	Cualquier entrada que termine en abc
abc . .	Cualquier entrada que comience por abc
. . abc . .	Cualquier entrada que contenga abc
. .	Cualquier entrada distinta de espacios en blanco
??abc	Cualquier entrada de cinco caracteres que termine por abc
<10	Cualquier entrada numérica menor que 10
>10	Cualquier entrada numérica mayor que 10
=10	Cualquier entrada numérica igual a 10
=1 . .10	Cualquier entrada numérica comprendida entre 1 y 10
/=10	Cualquier entrada numérica distinta de 10
/a . .	Cualquier entrada que no comience por a
/ . .	Cualquier entrada en blanco



El uso del PFS-File se resume en tres procesos habituales con los ficheros de tipo convencional: creación del modelo de ficha, introducción de datos en las diversas fichas y recuperación de datos de forma selectiva.

terminada revista, habrá que seleccionar, dentro del menú de funciones, la opción adecuada para "añadir información"; inmediatamente, aparecerá una página con el diseño inicial, aunque libre de datos. Ahora pueden ya introducirse los datos correspondientes a la revista que se desea incluir en el fichero. Esta operación puede prolongarse hasta ingresar todos los datos que sea preciso; por ejemplo, todas las revistas adquiridas en un mes.

Dentro de las funciones que brinda el PFS-File, existen varias destinadas a la búsqueda de datos. Al efecto, se llamará al formulario diseñado, y en alguno de los campos se indicará la característica de la búsqueda. Por ejemplo, si en el fichero de revistas se escribe "Enero-85" en el campo FECHA, y se dejan en blanco los restantes campos, la información de salida será la correspondiente a las revistas que se hayan adquirido con la mencionada fecha de publicación.

Las posibilidades de búsqueda que ofrece la aplicación, son lo suficientemente flexibles como para que sea factible obtener información de revistas sin conocer en su totalidad el contenido de uno de sus campos. Por ejemplo, puede ordenarse la localización de todas las revistas cuyo nombre comience por PROGRAM, sin más que teclear en el campo NOMBRE la siguiente expresión: PROGRAM.

En la figura adjunta se puede observar una lista completa de las posibilidades que brinda el paquete PFS-File para localizar información.

Para completar las posibilidades de extracción de datos, queda aún por mencionar la impresión parcial o total de ficheros en papel continuo. Dentro del menú de funciones del PFS-File se puede invocar a la denominada PRINT FORMS, adecuada para la escritura de datos a través de impresora.

Al seleccionarla, aparecerá un menú en la pantalla; en él habrá que decidir las opciones que se estimen oportunas: en qué dispositivo se producirá la escritura (puede ser una impresora o un disco), si se desea que en la escritura se incluyan los nombres otorgados en el diseño a cada dato, si se desea una pausa entre cada página, el número de líneas que se deben incluir en cada página y el número de copias que se desean obtener. Una vez especificados estos atributos, el programa producirá el informe del fichero que se encuentre activo.

Subrutinas

Del GOSUB al RETURN

En la terminología informática, una *rutina* es un bloque de instrucciones destinado a cumplir una misión específica dentro de un programa. La propia denominación, "rutina", sugiere tal definición. Y es que, realmente, el programa no es más que un encadenamiento de rutinas que se ejecutan una tras otra.

Suponga, por ejemplo, que se desea introducir un par de números a través del teclado, efectuar una operación matemática con ellos, y presentar el resultado en la pantalla. Cada una de estas acciones quedará encomendada a una rutina especializada, de tal forma que el programa para realizar tal actividad conjunta estará integrado por tres rutinas distintas.

Así pues, cabe concebir a las rutinas como módulos cuya ejecución da cuerpo a los programas. Su puesta en práctica hará que el programa en BASIC adquiera una estructura fácil de asimilar su cometido con una simple ojeada al listado. Además de contribuir a la claridad y organización funcional de un programa, y dar un aspecto limpio al listado del mismo, las rutinas tienen otras ventajas que aconsejan su empleo.

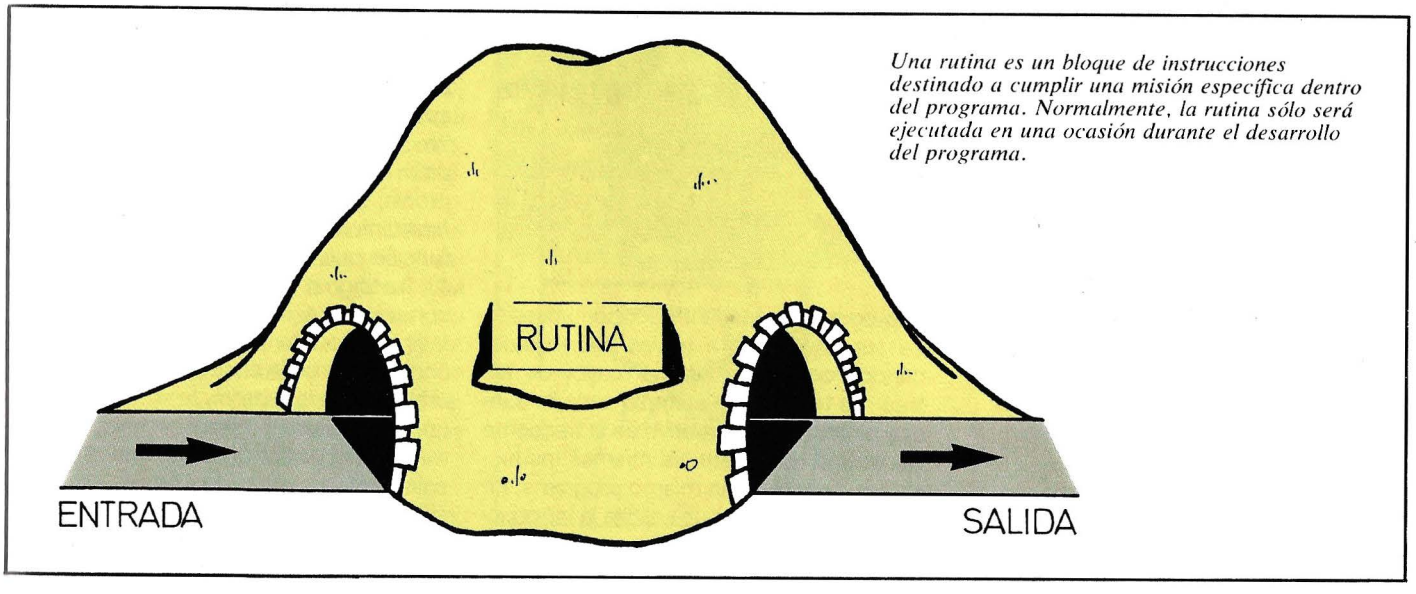
POR PURA RUTINA ...

En la estructura de cualquier programa cabe aún otra subdivisión, además de la ya establecida a partir de los módulos funcionales o rutinas. Aquí entra en liza la *subrutina*. De nuevo, una subrutina es un conjunto de instrucciones que forman un pequeño programa, anexo al programa principal, situado generalmente al final de éste (más adelante se verá el motivo).

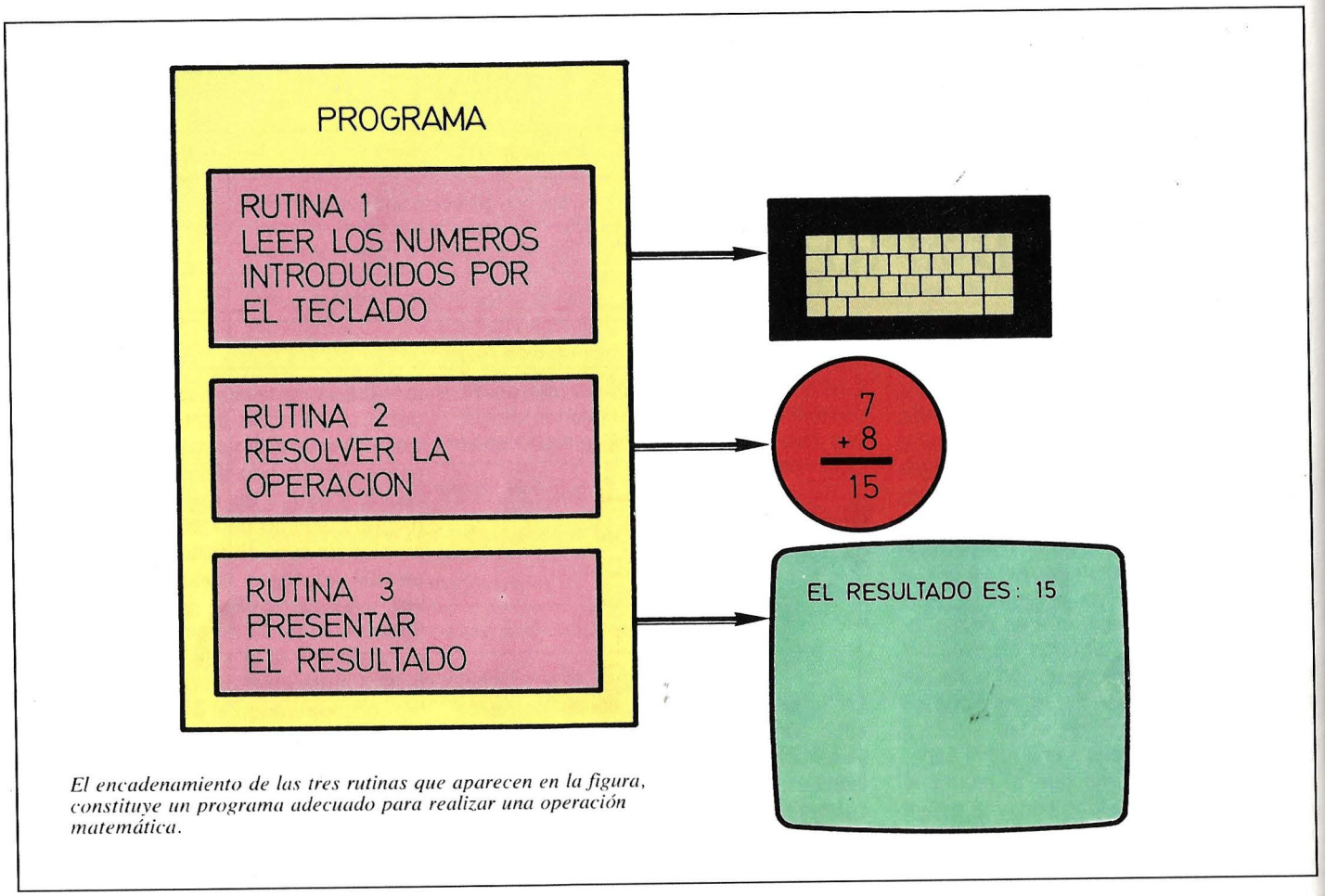
Las subrutinas —de menor entidad y volumen que las rutinas o módulos del programa— estarán constituidas por una serie de instrucciones, cuya ejecución se repetirá varias veces durante el desarrollo del programa. De inmediato saltan a la vista las primeras ventajas de las subrutinas: obvian la necesidad de repetir varias veces, y dentro del mismo programa, un mismo conjunto de instrucciones. Con el empleo de subrutinas sólo será necesario escribir las zonas coincidentes una sola vez, y acceder a ellas cuantas veces haga falta a lo largo del programa. Por lo tanto, es obvio que el uso de subrutinas contribuirá a rentabilizar la memoria del ordenador, economizando el espacio que ocuparían los bloques de instrucciones reemplazados por cada subrutina.

En este sentido se puede considerar a las subrutinas como "macroinstrucciones" disponibles en cualquier punto del programa.

En el sencillo ejemplo que acompaña a estas líneas, se puede observar cómo la repetición exhaustiva de las instrucciones que sirven para ingresar un número por el teclado, convierten al programa en monótono y repetitivo. Tal reiteración puede de-



Una rutina es un bloque de instrucciones destinado a cumplir una misión específica dentro del programa. Normalmente, la rutina sólo será ejecutada en una ocasión durante el desarrollo del programa.



El encadenamiento de las tres rutinas que aparecen en la figura, constituye un programa adecuado para realizar una operación matemática.

saparecer drásticamente apelando a una subrutina, como se verá más adelante.

```
10 REM EJERCICIO REPETITIVO
20 PRINT "INTRODUZCA UN NUMERO
   ENTRE 1 Y 100"
30 INPUT PRIMNUM
40 IF PRIMNUM<1 OR PRIMNUM>100
   THEN GOTO 20
50 PRINT "INTRODUZCA UN NUMERO
   ENTRE 1 Y 100"
60 INPUT SEGUNUM
70 IF SEGUNUM<1 OR SEGUNUM>100
   THEN GOTO 50
80 PRINT "INTRODUZCA UN NUMERO
   ENTRE 1 Y 100"
90 INPUT TERCNUM
100 IF TERCNUM<1 OR TERCNUM>100
   THEN GOTO 80
110 LET MAYOR=PRIMNUM
120 IF SEGUNUM>MAYOR THEN LET
   MAYOR=SEGUNUM
130 IF TERCNUM>MAYOR THEN LET
   MAYOR=TERCNUM
140 PRINT "EL NUMERO MAYOR ES:";MAYOR
150 END
```

La ejecución del programa dará el siguiente resultado en la pantalla:

```
INTRODUZCA UN NUMERO ENTRE 1 Y 100
?45
INTRODUZCA UN NUMERO ENTRE 1 Y 100
?83
INTRODUZCA UN NUMERO ENTRE 1 Y 100
?27
El NUMERO MAYOR ES: 83
■
```

El programa pide la introducción sucesiva de tres números a través del teclado, dando como resultado el mayor de los tres. Se trata de un ejemplo sencillo aunque elocuente e ilustrativo de la frecuente necesidad de repetir las mismas instrucciones dentro de un mismo programa. En el ejemplo, la zona que solicita la introducción de un número aparece en tres ocasiones.

Cuando un programa es excesivamente largo, el intento de seguir la pista a la secuencia de acciones programadas puede ser una tarea ardua e inacabable en muchos casos. Sobre todo cuando incluye frecuentes saltos o rupturas de secuencia, ya sean condicionales o incondicionales.

Desde luego, otra de las ventajas que aporta el uso de subrutinas es la de poder "modularizar" el programa. Permite dividirlo en pequeños módulos con entidad propia, que le darán una mayor claridad y facilitarán cualquier intento de descifrar la función que realiza.

Al fraccionar el programa en módulos compactos e independientes, será más fácil el detectar los posibles errores de codificación o de concepto que casi inevitablemente se producen en todo programa con una mínima complejidad. Así mismo, la corrección de un eventual error sólo afectará a una de las rutinas; e incluso será posible la sustitución completa de dicha rutina por otra, sin que ello afecte al funcionamiento del resto del programa.

Por último, otra de las ventajas que brinda el empleo de subrutina, es la de permitir su cómodo traslado de un programa a otro en el que sea necesario realizar la misma función que ejecutaba dicha rutina en el programa original.
Ello es posible debido a que la subrutina constituye un bloque independiente del programa principal, tal como si se tratara de un pequeño programa incluido en otro —el programa principal— de mayor entidad.
Semejante característica permite la creación de una biblioteca de subrutinas variadas que hará posible construir nuevos programas empleando rutinas de utilidad general, ya sea para obtener datos del teclado, almacenar datos en un archivo o, simplemente, crear una determinada presentación en pantalla.

UN VIAJE AL EXTERIOR

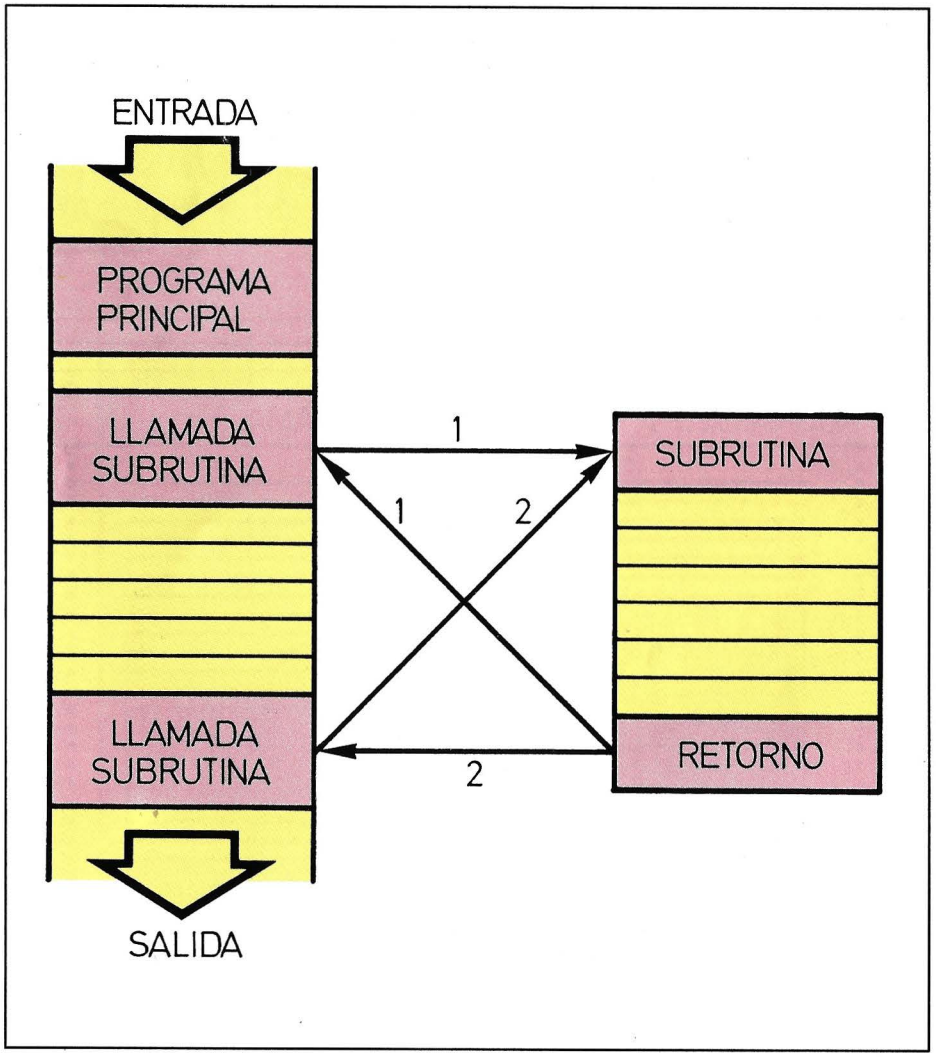
Llegados a este punto, cabe preguntarse cómo hay que construir las subrutinas y de qué forma se pueden utilizar dentro de un programa.
En principio, no hay que olvidar que todo programa sigue un cierto orden establecido en la ejecución de sus instrucciones; orden definido por los números de línea que preceden a las instrucciones del programa. La secuencia de ejecución puede ser desviada por las propias instrucciones. Por medio de instrucciones como GOTO, FOR/NEXT o IF/THEN, puede lograrse que el programa "salte" a otra zona, rompiendo el orden establecido por los números de línea.
Para que sea posible acceder a una subrutina será preciso desviar hacia ella la secuencia de ejecución del programa. Una vez ejecutado todo el conjunto de instrucciones de la subrutina, habrá que regresar al punto en el que se abandonó la ejecución del programa principal al producirse la llamada a la subrutina. ¡Un viaje de ida y vuelta!
El lenguaje BASIC cuenta con un comando especializado en este cometido; GOSUB (GO SUBrutine). Este comando se introducirá en cualquier punto del programa en el que se desee acceder a una subrutina.
El comando GOSUB debe ir precedido, como es habitual, por su correspondiente número de línea. Tras la palabra clave GOSUB se indicará el número de línea a partir del cual se encuentra situada la subrutina a la que se quiere acceder. Por consiguiente, el formato general de una instrucción GOSUB coincidirá con el siguiente:

<Núm.de línea> GOSUB <Núm.de línea de la subrutina>

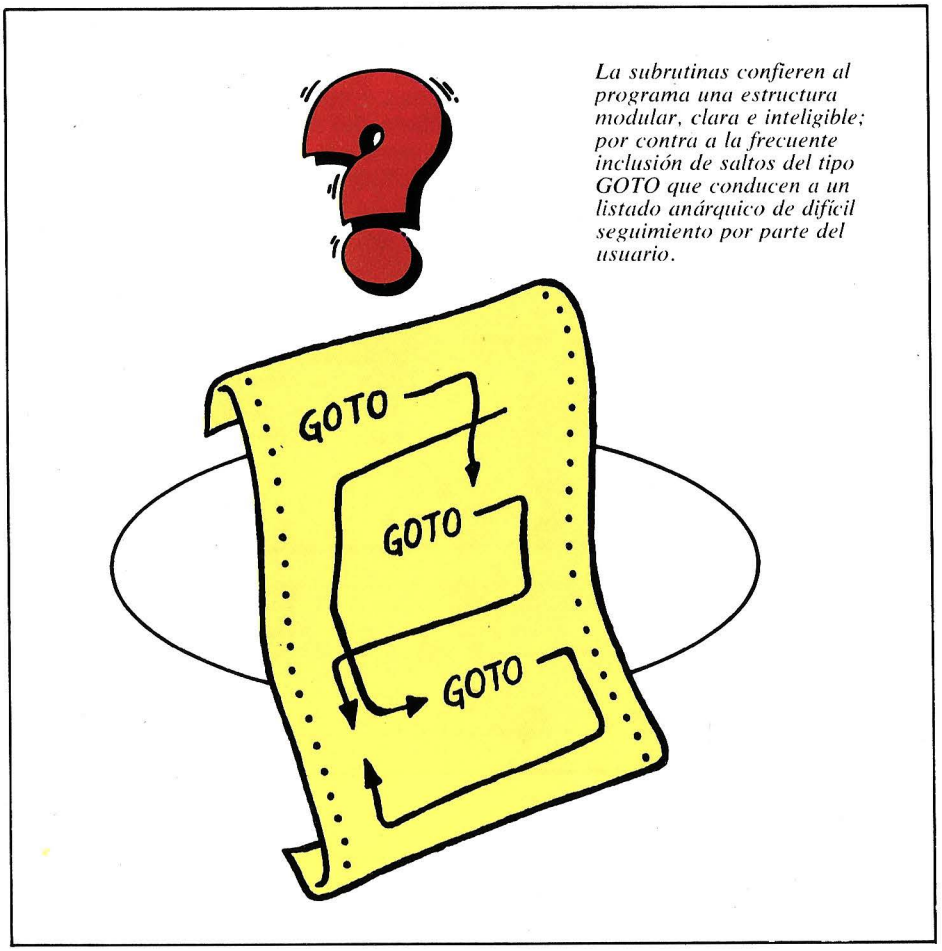
Por ejemplo:

100 GOSUB 2000

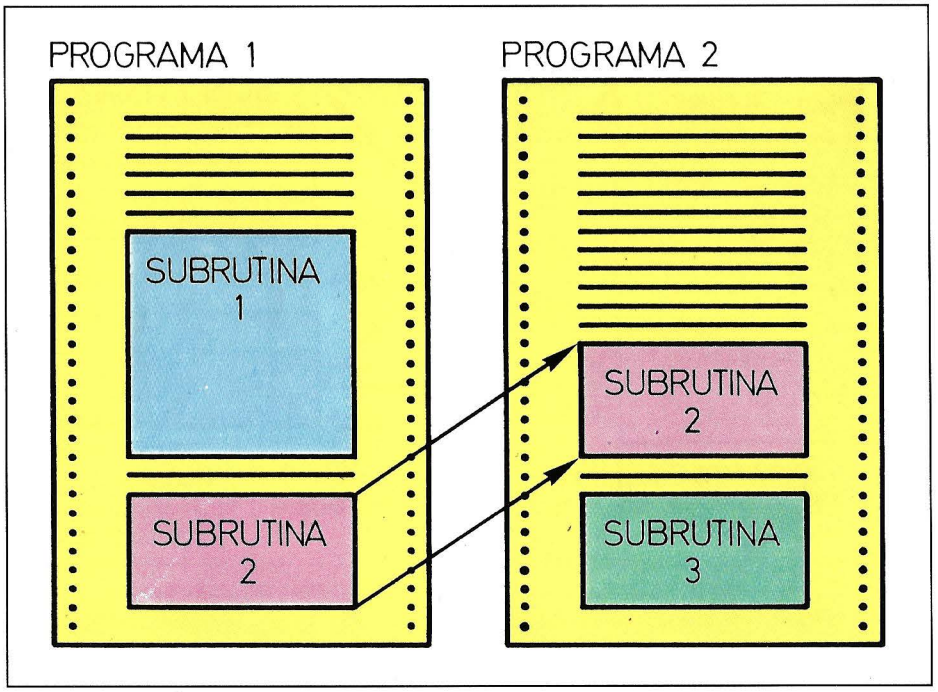
Cuando la ejecución del programa llegue a la línea que contiene la instrucción GOSUB, la secuencia saltará a la nueva línea cuyo número se indica. La ejecución continuará normalmente a partir de este punto del programa, coincidente con la entrada a la subrutina.
Una vez procesadas todas las instrucciones que conforman la subrutina, será preciso devolver la secuencia de ejecución al punto del programa donde se produjo la llamada a la subrutina; concretamente, a la línea inmediatamente siguiente a la que contiene la instrucción GOSUB.
Al llegar a una instrucción GOSUB, el microprocesador memoriza la dirección que corresponde a la instrucción que sigue a la de llamada a subrutina. De esta forma, conocerá con exactitud la dirección a la que debe de retornar.



Las subrutinas pueden compararse a pequeños programas, introducidos dentro de otro programa de mayor entidad, que pueden ser ejecutadas cuantas veces sea necesario.



La subrutinas confieren al programa una estructura modular, clara e inteligible; por contra a la frecuente inclusión de saltos del tipo GOTO que conducen a un listado anárquico de difícil seguimiento por parte del usuario.



Dada su propia organización y estructura, es posible utilizar una misma subrutina asociada a distintos programas.

EL TRAYECTO DE VUELTA

La orden de regreso al punto de origen se introduce como última instrucción de la subrutina. En el lenguaje BASIC la orden al efecto es RETURN. Al ejecutarla, el ordenador repara en que ha llegado al final de la subrutina, y por tanto que se debe "retornar" (RETURN) al punto de llamada. Su formato se reduce a la palabra clave precedida por el número de línea que le corresponda:

<Núm.de línea> RETURN

Por ejemplo:

```
2050 RETURN
```

El siguiente ejemplo contribuirá a clarificar la puesta en práctica de los comandos GOSUB y RETURN.

```
10 CLS
20 PRINT "SE EJECUTA EL PROGRAMA"
30 GOSUB 80
40 PRINT "Y OTRA VEZ"
50 GOSUB 80
60 PRINT "Y ASI ACABA EL PROGRAMA"
70 END
80 PRINT "ESTA ES LA SUBRUTINA"
90 RETURN
```

Su ejecución revela muy a las claras cuál es el cometido de las instrucciones de salto y retorno de subrutina.

```
SE EJECUTA EL PROGRAMA
ESTA ES LA SUBRUTINA
Y OTRA VEZ
ESTA ES LA SUBRUTINA
Y ASI ACABA EL PROGRAMA
■
```

A estas alturas, es posible ya confeccionar una alternativa al ejemplo propuesto al principio de este capítulo, utilizando ahora las subrutinas. El listado del nuevo programa puede adoptar el siguiente aspecto:

```
10 REM EJEMPLO UTILIZANDO SUBRUTINAS
15 LET MAYOR=1
20 FOR I=1 TO 3
30 GOSUB 70
40 NEXT I
50 PRINT "EL NUMERO MAYOR ES:";MAYOR
60 END
65 REM ...
70 REM SUBRUTINA
80 PRINT "INTRODUZCA UN NUMERO
   ENTRE 1 Y 100"
90 INPUT NUM
100 IF NUM<1 OR NUM>100 THEN GOTO 70
110 IF NUM>MAYOR THEN MAYOR=NUM
120 RETURN
```

No cabe duda que el programa ha ganado en claridad y en flexibilidad aunque, como se indicó anteriormente, no se trata más que de un simple ejemplo ilustrativo del uso de subrutinas dentro de un programa. Para que la diferencia entre ambos programas resulte del todo elocuente, cabe imaginar cuál será el aspecto de ambos si en lugar de buscar el mayor de tres números, se tratara de hallar el mayor de una colección de cien números introducidos por el teclado. En el segundo caso, empleando la subrutina sólo será necesario cambiar la línea 20 por la siguiente:

```
20 FOR I=1 TO 100
```

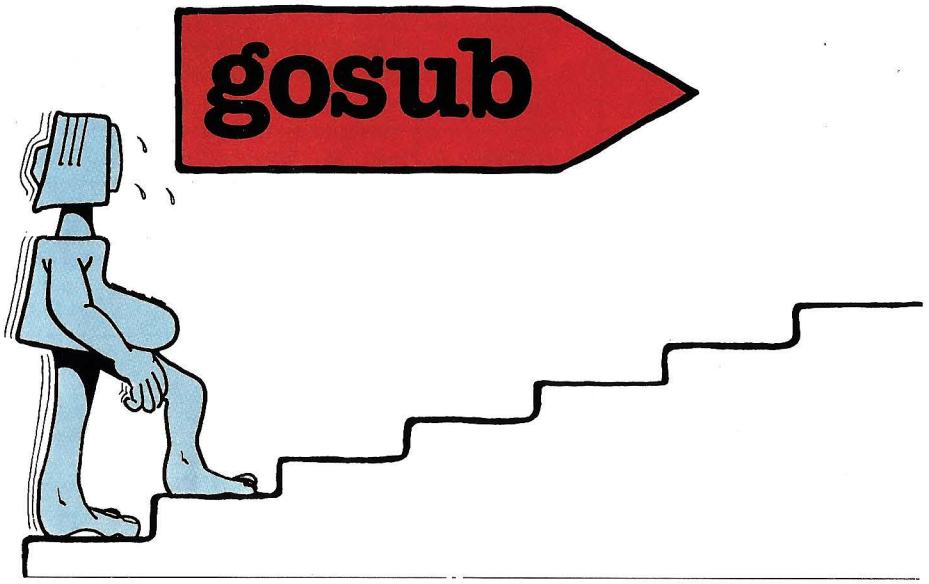
Sin embargo, de optar por el primer método, el programa listado al principio habría que completarlo con varios cientos de líneas. La ventaja que supone el uso de subrutinas queda, pues, fuera de cualquier posible duda.
Como es fácil imaginar, toda subrutina debe concluir con una instrucción RETURN. Sólo de esta forma será posible regresar al punto de llamada. De lo contrario, al finalizar la subrutina sin el correspondiente RETURN, la máquina emitirá un mensaje de error.
En otro orden, si durante la ejecución del programa aparece una instrucción RETURN, sin que su presencia esté motivada por la llamada a una subrutina con el comando GOSUB, se generará el correspondiente mensaje de error.

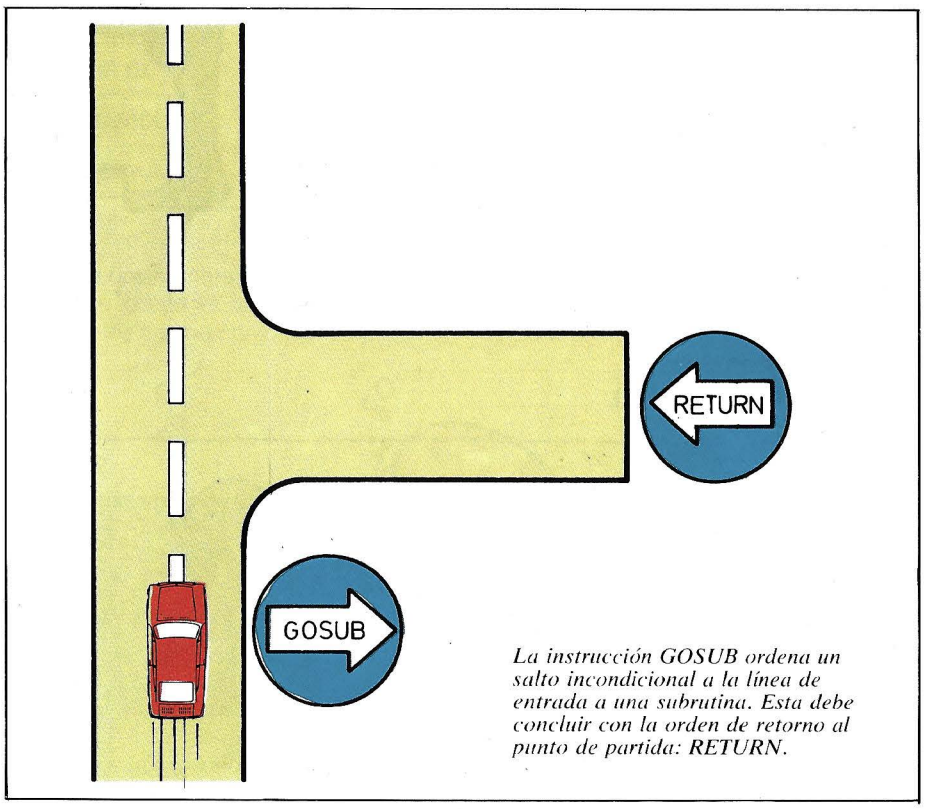
GOSUB

Transfiere el control del programa a la subrutina cuyo número de línea inicial se especifica.

Formato: <*NI*> *GOSUB* [<*nl*> <*expresión*>]

Ejemplos: 10 GOSUB 1000
50 GOSUB A+100





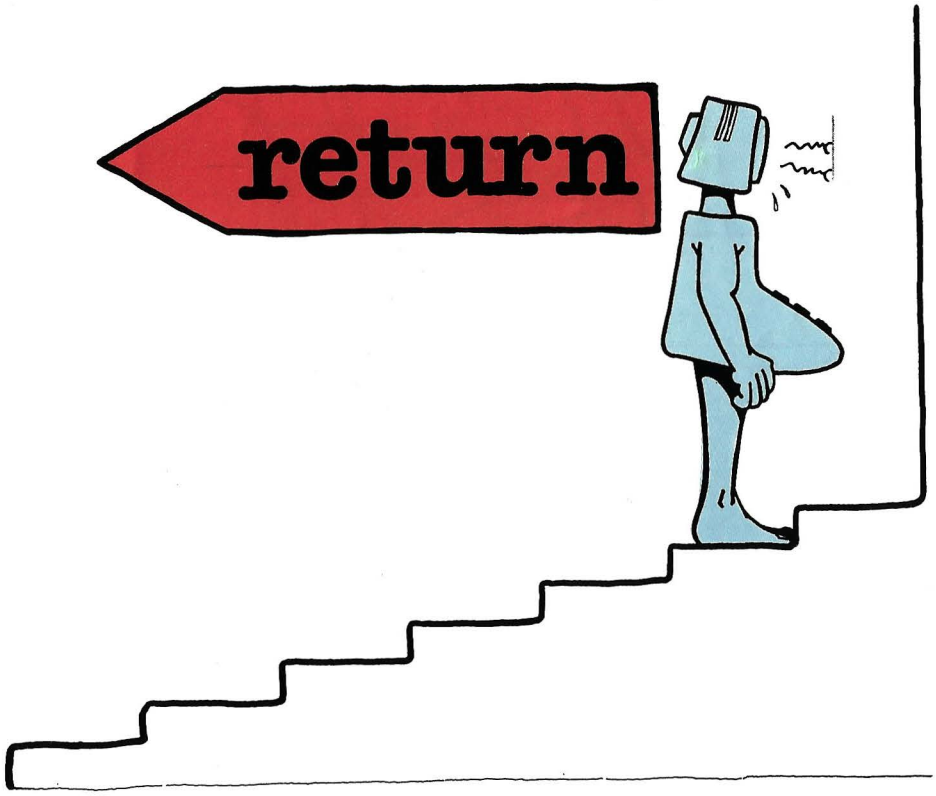
La instrucción GOSUB ordena un salto incondicional a la línea de entrada a una subrutina. Esta debe concluir con la orden de retorno al punto de partida: RETURN.

Para evitar este tipo de errores es conveniente clarificar la estructura del programa. Con este objetivo, suelen emplazarse las distintas subrutinas detrás de la última instrucción del programa principal que, generalmente, será una instrucción END. De todas formas, ello no es obligatorio.

ANIDAMIENTO DE SUBRUTINAS

Las subrutinas constituyen un recurso inapreciable para el programador. Basta con pensar que una simple rutina puede ser llamada en innumerables ocasiones, evitando la obligación de escribir repetidamente múltiples bloques del programa con idénticas instrucciones. Al igual que ocurría con los bucles, también las subrutinas son anidables, unas dentro de otras. Ello significa que durante la ejecución de una subrutina es posible efectuar una llamada a otra subrutina, y de esta última también es posible saltar de nuevo a otra distinta y así sucesivamente.
El número máximo de anidamientos permisibles depende de cada equipo; y más exactamente del tamaño de la "pila" o zona de memoria destinada a memorizar las direcciones de retorno. Al anidar subrutinas, la referida "pila" irá almacenando tantas direcciones como anidamientos haya, para luego gestionar el retorno a cada una de ellas por orden inverso al de llamada.
El siguiente listado corresponde a un programa adecuado para clasificar nombres alfabéticos.

```
 10 REM CLASIFICADOR
 20 INPUT "CUANTOS NOMBRES DESEA
    CLASIFICAR";N
 30 DIM N$(N)
 40 FOR I=1 TO N
 50 INPUT "INTRODUZCA NOMBRE",N$(I)
 60 NEXT I
 70 GOSUB 120
 80 FOR I=1 TO N
 90 PRINT N$(I)
100 NEXT I
110 END
115 REM ...
120 REM COMPROBAR ORDEN
130 LET YA=0
140 FOR K=1 TO N-1
150 IF N$(K)>N$(K+1) THEN GOSUB 190
160 NEXT K
170 IF YA=0 THEN RETURN
180 GOTO 130
185 REM ...
190 REM COLOCAR
200 LET A$=N$(K)
210 LET N$(K)=N$(K+1)
220 LET N$(K+1)=A$
230 LET YA=1
240 RETURN
```



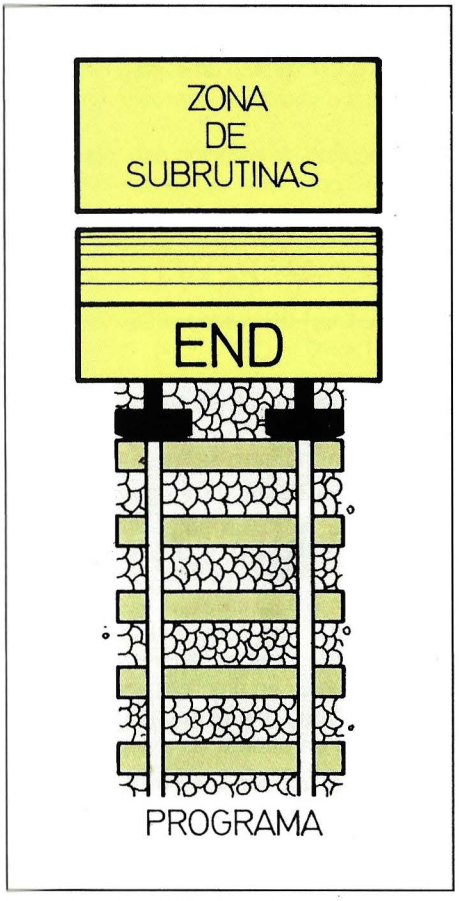
RETURN

Señala el final de una subrutina; devuelve el control del programa a la siguiente instrucción a la que efectuó la llamada.

Formato: <NL> RETURN

Ejemplo: 100 RETURN

Al analizar las líneas que lo componen, se observa que en él existe un anidamiento de subrutinas. La primera llamada a subrutina se encuentra en la línea 70. A su vez, cuando es necesario, por efecto de la condición impuesta en la línea 150, se produce una nueva llamada a la subrutina localizada a partir de la línea 190.
Analicemos con más calma cuál es el funcionamiento del programa.
En la línea 20 se pide el número de nombre (N) que se quieren introducir para su clasificación. Este número se utilizará posteriormente en distintas zonas del programa. Dicho número no está limitado inicialmente a ningún valor; sin embargo, habrá que tener en cuenta la limitación impuesta por el espacio de memoria que el equipo pone a disposición del usuario. Otro factor a considerar es el número de líneas visualizables simultáneamente en la pantalla. Este detalle no se ha previsto en el programa. Si sólo caben 24 líneas en la pantalla, sólo se visualizarán de forma estable los últimos veinticuatro nombres clasificados; los anteriores desaparecerán



La instrucción END (fin del programa) aporta la barrera infranqueable en la que termina la secuencia principal de un programa BASIC. Tras ella pueden emplazarse las subrutinas del programa que, de esta forma, quedarán resguardadas de cualquier ejecución no deseada.

TABLA DE CONVERSION

ORDENADOR	GOSUB	RETURN	ON/GOSUB
	GOSUB <nl>	**RETURN**	**ON <exp.> GOSUB <nl>**
APPLE II (APPLESOFT)	GOSUB <nl>	RETURN	ON <exp.> GOSUB <nl>
APRICOT (M-BASIC)	GOSUB <nl>	RETURN	ON <exp.> GOSUB <nl>
ATARI	GOSUB <nl>	RETURN	ON <exp.> GOSUB <nl>
CBM 64	GOSUB <nl>	RETURN	ON <exp.> GOSUB <nl>
DRAGON	GOSUB <nl>	RETURN	ON <exp.> GOSUB <nl>
EQUIPOS MSX	GOSUB <nl>	RETURN	ON <exp.> GOSUB <nl>
HP-150	GOSUB <nl>	RETURN	ON <exp.> GOSUB <nl>
IBM PC	GOSUB <nl>	RETURN	ON <exp.> GOSUB <nl>
MPF	GOSUB <nl>	RETURN	ON <exp.> GOSUB <nl>
NCR DM-V (MS-BASIC)	GOSUB <nl>	RETURN	ON <exp.> GOSUB <nl>
NEW BRAIN	GOSUB <nl>	RETURN	ON <exp.> GOSUB <nl>
ORIC	GOSUB <nl>	RETURN	ON <exp.> GOSUB <nl>
SHARP MZ-700 (MZ-BASIC)	GOSUB <nl>	RETURN	ON <exp.> GOSUB <nl>
SINCLAIR QL	GOSUB <nl>	RETURN	ON <exp.> GOSUB <nl>
SPECTRAVIDEO	GOSUB <nl>	RETURN	ON <exp.> GOSUB <nl>
ZX-SPECTRUM	GOSUB <nl>	RETURN	—

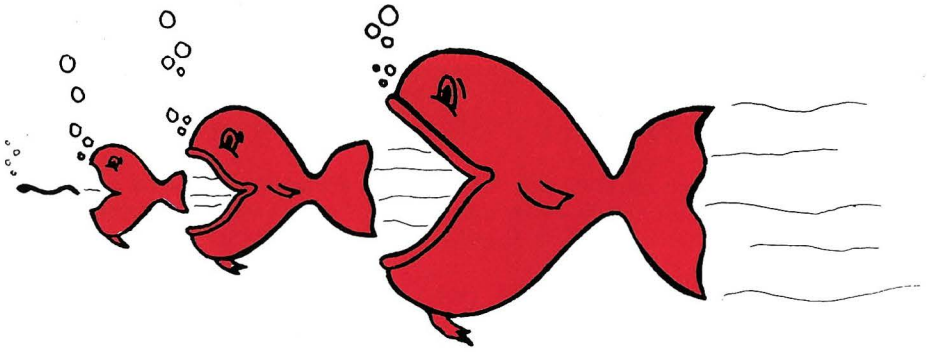
<nl>: Número de línea. <exp.>: Expresión o variable numérica.

por efecto del desplazamiento vertical o "scroll", empujados hacia el borde superior de la pantalla por los sucesivos nombres.

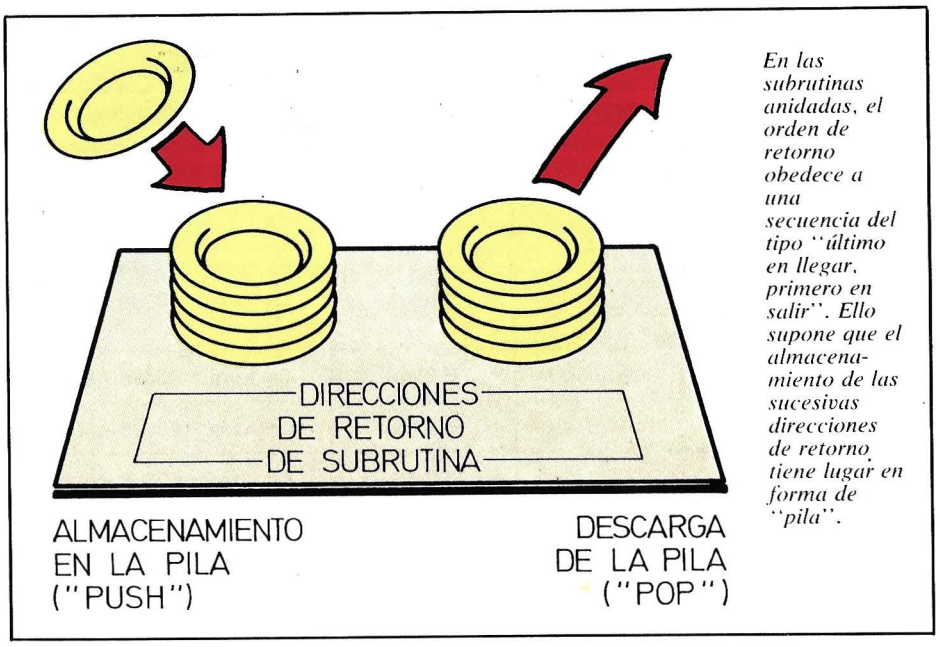
En la línea 30 se dimensiona la matriz N$ para dar cabida a los N nombres. Estos se introducirán al ejecutarse el bucle constituido por las tres líneas siguientes del programa.

La línea 70 hace una llamada a la subrutina que comienza en la línea 120, cuya estructura puede parecer un tanto extraña a primera vista. La última instrucción no es un simple RETURN, como ha sido habitual hasta el momento, sino que dicha orden aparece en el argumento de la instrucción 170. En efecto, la salida de dicha subrutina sólo será efectiva una vez que se haya cumplido la condición YA=0, o lo que es lo mismo: cuando se hayan terminado de clasificar todos los nombres.

La clasificación se hace comprobando, para cada dos nombres consecutivos, cuál es el que ocupa el orden inferior. Si el orden de los dos nombres comparados no es el correcto, se ejecutará zona THEN de la línea 150. Esta coincide con una orden de salto a otra subrutina cuya misión es

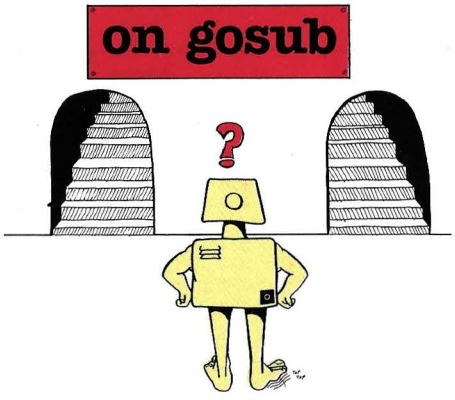


El anidamiento de subrutinas se produce al llamar a una nueva subrutina durante la ejecución de otra subrutina de nivel superior. La segunda subrutina, a su vez, puede llamar a una tercera y así sucesivamente.



En las subrutinas anidadas, el orden de retorno obedece a una secuencia del tipo "último en llegar, primero en salir". Ello supone que el almacenamiento de las sucesivas direcciones de retorno tiene lugar en forma de "pila".

permutar el orden de ambos nombres. A su vez, la misma subrutina pone a 1 el indicador YA, el cual identifica que se ha producido un cambio de orden.
Si el identificador YA vale 1 al concluir cada tanda de comparación de todos los nombres, se tendrá la certeza de que se ha efectuado el cambio entre al menos dos nombres. Por lo tanto, será necesario inicializarlo a 0 y volver a comprobar de nuevo si el orden es correcto. En tal caso —si el orden es correcto— la secuencia de ejecución será devuelta al programa principal, quien se encargará de mostrar en pantalla los nombres, finalizando así el programa.



SALTO CONDICIONAL A SUBRUTINAS

El término medio entre las instrucciones condicionales (IF/THEN/ELSE) y las transferencias temporales de control (GOSUB) aparece con la posibilidad de elegir la subrutina que debe ejecutarse, dependiendo del resultado de evaluar una determinada condición o expresión matemática.
Esta facultad de elección está prevista en el BASIC por medio del comando ON/GOSUB. Su formato es totalmente análogo al del comando ON/GOTO estudiado en un capítulo anterior:

(Núm.de línea) ON <exp.> GOSUB <n11>[,<n12>, ...]

Esta instrucción calcula el valor entero de la expresión <exp.> y llama a la subrutina cuyo primer número de línea es <n11> si el valor calculado es 1, a la subrutina <n12> si el valor calculado es 2, y así sucesivamente hasta el último número de línea especificado detrás de la palabra GOSUB.
Estas instrucciones son profusamente utilizadas por los programadores más versados. Su puesta en práctica permite conferir al programa una estructura organizada por medio de subrutinas independientes, lo que contribuye a mejorar su claridad y a facilitar la tarea de programación y puesta a punto del mismo.
Una aplicación típica de la instrucción ON/GOSUB es la de seleccionar entre un conjunto de opciones a través de un menú. Ello puede realizarse con una estructura semejante a la que sigue:

```
10 PRINT "MENU"
20 PRINT
30 PRINT "1.ENTRADA DE DATOS"
40 PRINT "2.CALCULO DE VALORES"
50 PRINT "3.ALMACENAMIENTO DE DATOS"
60 PRINT "4.SALIDA POR IMPRESORA"
70 INPUT N
80 ON N GOSUB 100,200,300,400
90 END
...
100 REM SUBRUTINA DE ENTRADA
...
200 REM CALCULOS
...
300 REM ALMACENAMIENTO
...
400 REM IMPRESION
...
```

Si el valor de N no está dentro del margen contemplado por la instrucción ON/GOSUB (esto es: N<1 ó N>4 en el ejemplo), la ejecución continuará en la línea siguiente, sin que sea invocada ninguna subrutina.

ON/GOSUB

Transfiere el control a una subrutina de forma condicional y selectiva. El salto se producirá tras evaluar la expresión que ocupa la zona ON; concretamente, a la subrutina cuyo número de línea ocupa, en el argumento de GOSUB, la posición señalada por el resultado de la expresión.

Formato: <Nl> ON <exp.> GOSUB <nl1>[,<nl2>,...,<nlk>]

Ejemplos:
```
10 ON X GOSUB 1000, 2000
80 ON X+10-B GOSUB 100,200,300,400
```

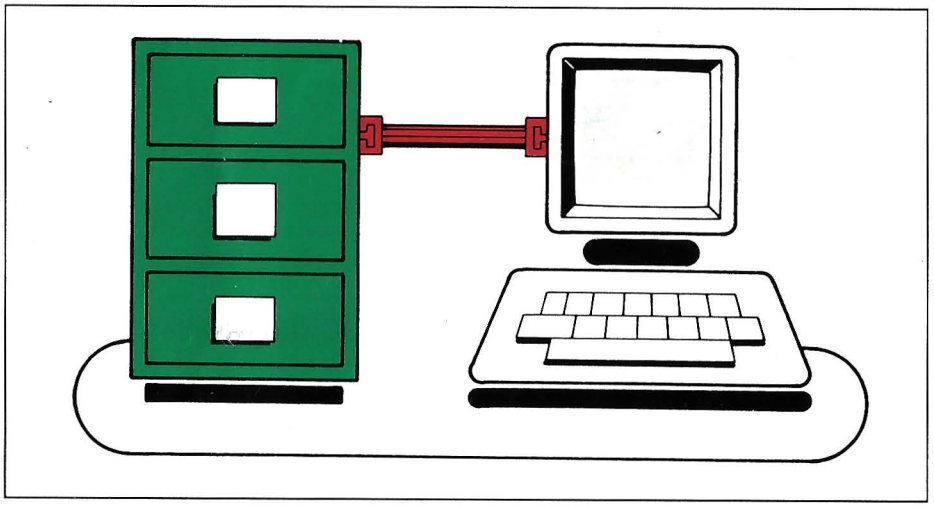
Logo (14)

Archivos y primitivas

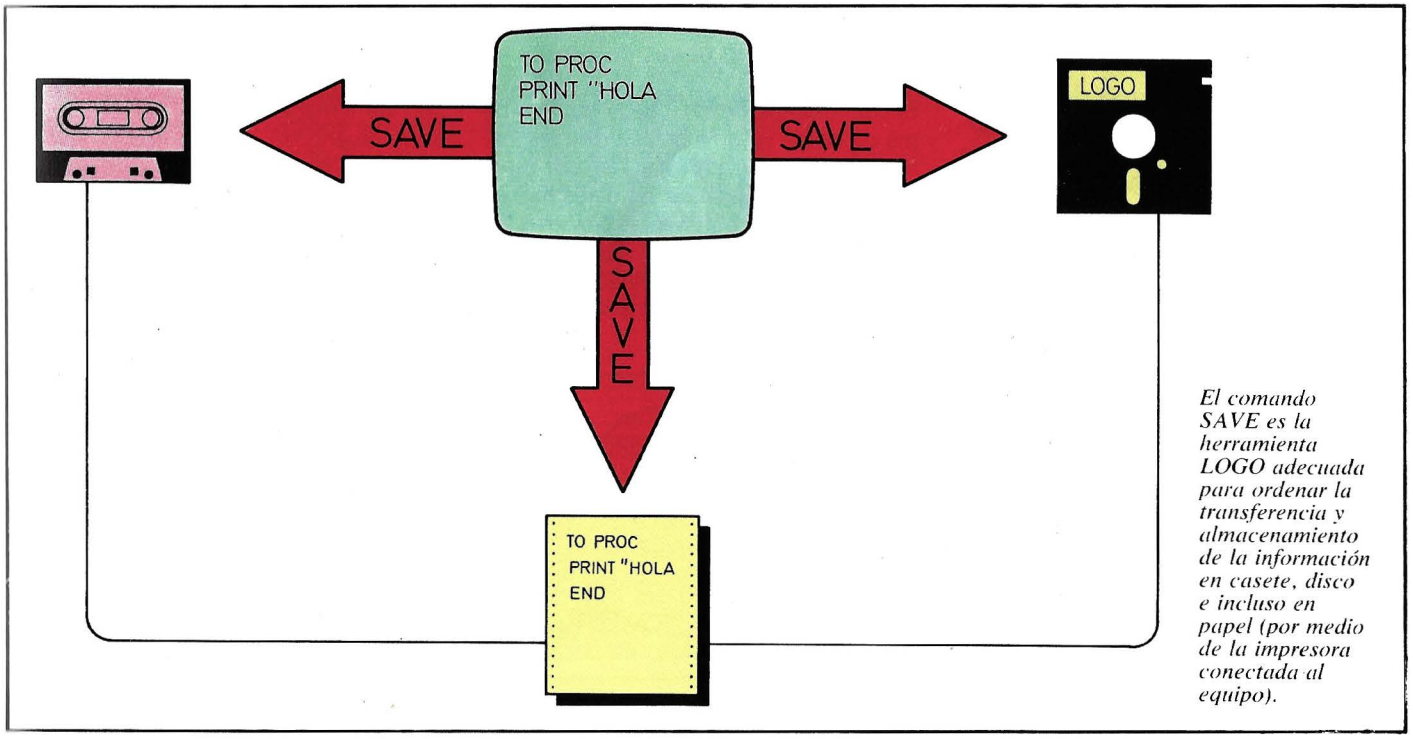
El entorno que rodea al ordenador incluye todo un abanico de dispositivos periféricos. Entre ellos se encuentran las impresoras, unidades de disco, de casete, etc. En los dispositivos dedicados al almacenamiento masivo (cintas y discos) se suele agrupar la información en "archivos".

Un archivo es un conjunto homogéneo de datos manipulable bajo un nombre genérico. Los archivos son, pues, estructuras de datos.

El primer ejemplo surge con el deseo de conservar los procedimientos creados en una sesión de trabajo. Es sabido que al desconectar el aparato el contenido de su memoria se pierde. Para evitar la repetición del mismo trabajo, hay que hacer uso de

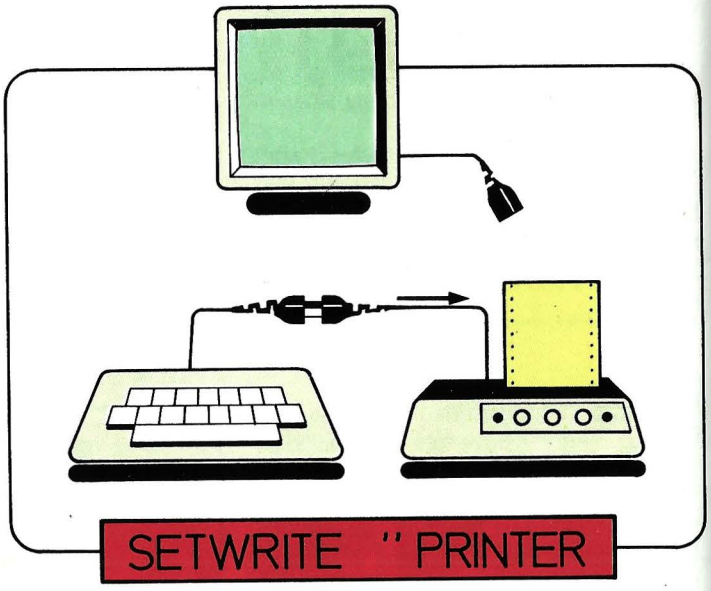


En el lenguaje LOGO, el almacenamiento de la información en las memorias de masa se realiza en forma de archivos.



El comando SAVE es la herramienta LOGO adecuada para ordenar la transferencia y almacenamiento de la información en casete, disco e incluso en papel (por medio de la impresora conectada al equipo).

Los datos de entrada pueden tomarse de un dispositivo periférico distinto al teclado, por ejemplo de una unidad de disco. El comando adecuado para seleccionar tal posibilidad es SETREAD.



El uso del comando SETWRITE permite canalizar los datos de salida hacia otro periférico distinto de la pantalla.

los periféricos de almacenamiento permanente. Los más populares son la cinta y el disco magnético. Con ellos es posible preservar la información contenida en el espacio de trabajo.

El comando que permite archivar la información es SAVE. Este necesita un dato de entrada que será una palabra identificativa del archivo creado. En un mismo soporte (cinta en casete o disco) se pueden guardar varios archivos. Este es el motivo por el cual es preciso diferenciarlos con distintos nombres.

En el caso de que exista más de un dispositivo de almacenamiento conectado al ordenador, es preciso indicar cuál de ellos debe memorizar el archivo. La identificación de los periféricos suele diferir de uno a otro equipo; aunque, con frecuencia, se suele recurrir a las siguientes letras: C (casete), D (unidad de disco) y P (impresora).

Si el periférico elegido es la impresora, se obtendrá una copia en papel del contenido del espacio de trabajo.

El comando que realiza la operación contraria es LOAD. Este carga en memoria el archivo especificado por su dato de entrada. Así, un espacio de trabajo grabado con SAVE "TORTU1, se puede recuperar con LOAD "TORTU1.

El comando LOAD no puede ser utilizado cuando el periférico elegido es la impresora. Es obvio que este dispositivo acepta datos, aunque no los devuelve al ordenador.

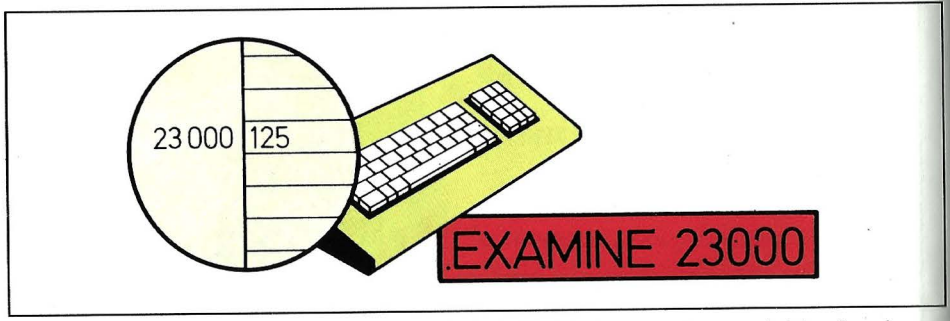
MANEJO DE ARCHIVOS

El manejo de archivos no se limita al almacenamiento de procedimientos. Es posible guardar y recuperar datos (numéricos, palabras y/o listas). Para ello se recurre a permutar los dispositivos de entrada/salida.

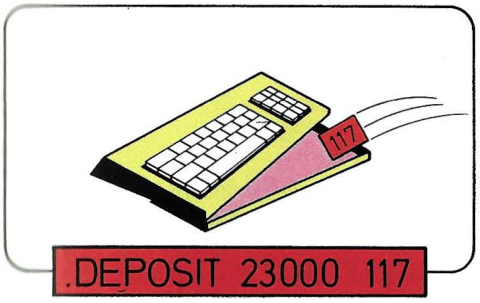
El comando SETWRITE hace que los datos de salida vayan a parar a otro dispositivo distinto de la pantalla. Así, por ejemplo, es posible canalizar los datos hacia la impresora, para obtener resultados fácilmente manejables. El siguiente procedimiento hará que los resultados de ejecutar el procedimiento PRO1 salgan a través de la impresora.

```
TO IMPRE
SETWRITE "P:
PRO1
SETWRITE [ ]
END
```

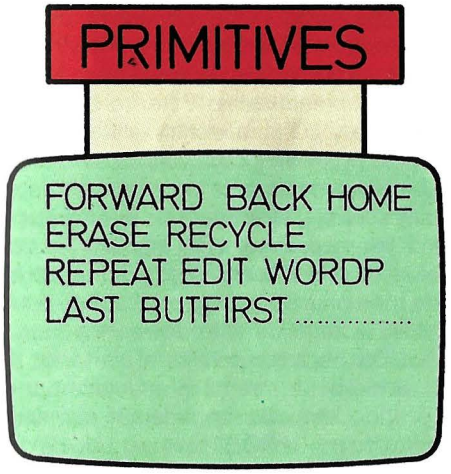
El periférico de salida puede ser una unidad de almacenamiento, de disco o casete, creándose, de esta forma, archivos de datos no volátiles. De esta forma al ejecutarse una orden PRINT o similar (TYPE, SHOW ...) el dato no aparecerá en pantalla, sino que será escrito en el archivo externo. Para volver a activar la pantalla bastará con ejecutar un comando SETWRITE [ ] (cadena vacía).



La función ".EXAMINE" permite entrar en la intimidad de la memoria central del ordenador: revela el contenido de la posición de memoria que se indique como dato de entrada.



El comando ".DEPOSIT" se encarga de depositar un dato en la posición de memoria que se indique.



Otra de las posibilidades del LOGO se concreta en el examen de todas las palabras reservadas. Para obtener un listado de todas ellas hay que hacer uso del comando ".PRIMITIVES".

TABLA DE ORDENES-LOGO

INSTRUCCION	COMETIDO	OPERADOR/ COMANDO
SAVE <arch>	Guarda el espacio de trabajo en el archivo indicado	Comando
LOAD <arch>	Recupera un espacio de trabajo del archivo indicado	Comando
SETWRITE <arch>	Canaliza los datos de salida hacia el archivo indicado (en lugar de hacia la pantalla)	Comando
SETREAD <arch>	Hace que se recojan los datos de entrada del archivo indicado (en lugar de obtenerlos del teclado)	Comando
CATALOG	Muestra los nombres de los archivos que residen en el disco	Comando
ERF <arch>	Borra del disco el archivo indicado	Comando
.PRIMITIVES	Muestra todas las palabras reservadas del LOGO	Comando
.EXAMINE <pos>	Devuelve el contenido de una posición de memoria	Operador
.DEPOSIT <pos> <N>	Introduce el dato <N> en la posición de memoria especificada	Comando
.CALL <pos>	Ejecuta la rutina en código máquina situada en la posición de memoria especificada	Comando

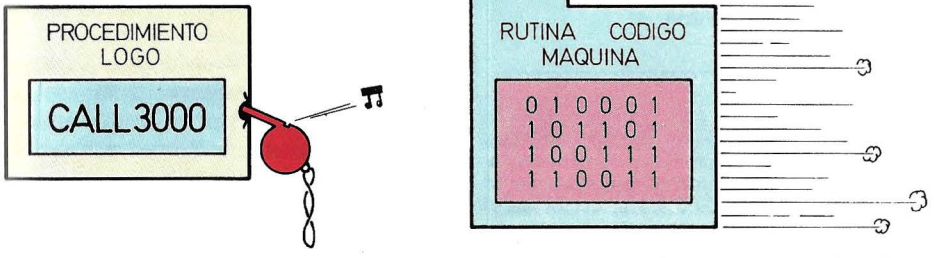
<arch>: Nombre de archivo; junto al mismo hay que indicar el dispositivo en el que se encuentra. <pos>: Número que identifica una posición de memoria del ordenador. <N>: Dato numérico a situar en una posición de memoria. Ha de ser entero, positivo e inferior a 256.

El método para recuperar un archivo pasa por permutar el teclado con dicho archivo. Esta acción se define indicando el archivo de entrada en un comando SETREAD. Una vez seleccionado el archivo, cada vez que se ejecute una orden RC o RL los datos se tomarán del mismo, en lugar del teclado. La devolución del control al teclado se realiza por medio de la orden SETREAD [ ].

Existen otros dos comandos específicos para la operación con disco magnético. Estos son CATALOG y ERF.
El primero proporciona una tabla que relaciona todos los archivos contenidos en el disco, mientras que ERF borra archivos (ERase File). Este último comando eliminará el archivo cuyo nombre coincida con la palabra incluida como dato de entrada de ERF.

Los dos procedimientos que se relacionan a continuación operan asociados: gestionan el almacenamiento en disco y la emisión de un listado por impresora del procedimiento LOGO que se indique.

```
TO GRABA :P
MAKE "DIS WORD "D: :P
SAVE :DIS
SETREAD :DIS
SETWRITE "P:
LISTADO
SETREAD [ ]
SETWRITE [ ]
END

TO LISTADO :A
MAKE "L RL

IF EMPTYP :L [STOP]
PRINT :L
LISTADO
END
```



El cometido de ".CALL" es transferir el control a una rutina en código máquina. Su dato de entrada especifica la posición de memoria a partir de la que está situada la rutina en cuestión.

PALABRAS CLAVE DEL LOGO

INGLES	CASTELLANO	INGLES	CASTELLANO
TS	MODOTEXTO	PC	COLOR, COLORLAPIZ
COUNT	CUENTA	SETH	PONRUMBO
BUTFIRST	MENOSPRIMERO	XCOR	XCOOR
FIRST	PRIMERO	YCOR	YCOOR
BUTLAST	MENOSULTIMO	HEADING	RUMBO
LAST	ULTIMO	POS	DONDE, POSICION
FPUT	PONP, COLOCAPRI	SETSP	PONVELOCIDAD
LPUT	PONU, COLOCAULT	SPEED	VELOCIDAD
SE	ORACION, FRASE	WINDOW	VENTANA
NUMBERP	NUMERO?	WRAP	ENRROLLA
LISTP	LISTA?	TELL	DECIR
WORDP	PALABRA?	WHO	QUIEN
WORD	PALABRA	EACH	CADA
EMPTYP	VACIA?	ROUND	REDONDEA
MEMBERP	PERTENECE?	INT	ENT
EQUALP	IGUAL?	REMAINDER	RESTO
CHAR	ESCRIBECARACTER, CARACTER	RANDOM	AZAR
		SUM	SUMA
ASCII	NUMCARACTER, ASCII	PRODUCT	PRODUCTO
MAKE	HACER, ASIGNA	SIN	SENO
NAMEP	NOMBRE?	COS	COSENO
THING	COSA, ITEM	SQRT	RAIZCUADRADA
EDIT	EDITA, REVISAR	QUOTIENT	COCIENTE
EDNS	EDNS	TRUE	CIERTO
TO	PARA	FALSE	FALSO
END	FIN	AND	AMBOS, Y
ERNS	SUPNS	OR	UNOUOTRO, O
ERALL	SUPTODO	NOT	NO
ERPS	SUPPS	IF	SI
ERASE	SUPRIME	STOP	PARATE, ALTO
ERN	SUPN	OUTPUT	RESPUESTA, DEVUELVE
POALL	ET, ESCTODO	RUN	CUMPLE, EJECUTA
POTS	EP, ESCTS	REPEAT	REPETIR, REPITA
PO	EI, ESCPROC	TYPE	ESCRIBE
POPS	ESCPROCS	PRINT	IMPRIME
PONS	EN, ESCNS	SHOW	MUESTRA
RECYCLE	RECICLA	SETCURSOR	PONCURSOR
NODES	NODOS	RC	LEECARACTER, LEECAR
BACK	ATRAS, RETROCEDE	RL	LEELINEA, LEELISTA
FORWARD	ADELANTE	KEYP	LC?, TECLA?
CS	BORRAPANTALLA	BEEP	SONIDO
CLEAN	BORRA, LIMPIA	NOBEEP	SILENCIO
RIGHT	DERECHA, GIRADERECHA	LOAD	RECUERDA, CARGA
LEFT	IZQUIERDA, GIRAIZQUIERDA	CATALOG	CATALOGO
		SAVE	GUARDA
HOME	CENTRO	ERF	ELIMINAARCHIVO
PX	PLUMAINVERSA, LAPIZREVES	.PRIMITIVES	.PRIMITIVAS
		.EXAMINE	.EXAMINA
PE	PLUMADEBORRAR, GOMA	.DEPOSIT	.DEPOSITA
PD	CONPLUMA, CONLAPIZ	.CALL	.LLAMA
PU	SINPLUMA, SINLAPIZ		
HT	OCULTATORTUGA		
ST	MUESTRATORTUGA		
SHOWNP	VISIBLE?		
SETBG	COLORFONDO, PONCF		
BG	FONDO, COLORFONDO		
SETX	PX, PONX		
SETY	PY, PONY		
SETPOS	PXY, PONPOS		
SETPC	PONCOLOR, PONCL		

Existen algunas versiones del lenguaje LOGO cuyo vocabulario de palabras reservadas se encuentra en castellano. Aunque los nombres no coinciden en todas las versiones, las más utilizadas son las que se exponen en la tabla. Las palabras clave se relacionan en el mismo orden en el que se han estudiado a lo largo de los capítulos de la obra dedicados al LOGO.

PRIMITIVAS ESPECIALES

En LOGO se denomina primitiva a toda palabra clave o palabra reservada. Estas coinciden con los comandos y operadores propios del LOGO. Algunas primitivas tienen un carácter especial; éstas son las que van a constituir el objeto de los próximos párrafos.

Las primitivas especiales se caracterizan porque su nombre comienza por un punto. La más inmediata es .PRIMITIVES cuya misión es la de mostrar todas las palabras reservadas. Este comando resulta muy útil para identificar las palabras claves propias de cada versión del LOGO.

Un grupo destacable de primitivas especiales es el que acoge a las que afectan a las "interioridades" de la máquina. Dos de ellas se utilizan para acceder a posiciones de memoria. Por ejemplo, la función .EXAMINE admite un dato de entrada numérico y entrega como salida el contenido de la posición de memoria apuntada por el referido dato de entrada. El comando opuesto es .DEPOSIT que se nutre de dos datos de entrada. El primero determina la posición de memoria y el segundo el dato a almacenar en la misma.

Con estas primitivas y un ligero conocimiento del código máquina se pueden crear rutinas en dicho lenguaje. Para encadenar rutinas (creadas por éste u otro método) con procedimientos LOGO, se dispone de la orden .CALL (llamar). Su efecto no es otro que el de ejecutar la rutina en código máquina situada a partir de la posición de memoria especificada. La posición se indica como dato numérico de entrada a la orden .CALL.

Como última primitiva cabe mencionar a .SETSCR (de SET SCReen, poner pantalla). Este comando define la relación entre las posiciones horizontales y verticales, lo que equivale a fijar la relación entre la resolución vertical y horizontal de la pantalla.

Utilizando .SETSCR de forma adecuada será posible ejecutar la representación en pantalla para evitar, por ejemplo, que los cuadrados aparezcan como rectángulos. El problema cuya resolución se encomienda a .SETSCR, deriva de las diferencias que surgen en la visualización de un mismo trazado por parte de distintos receptores de TV domésticos.

MS/DOS (2)

Descripción general del DOS 1.0

La versión DOS 1.0 del sistema operativo MS/DOS está destinada a la gestión de archivos en disco flexible (disquete). Por contra la versión DOS 2.0, más avanzada, se orienta a la gestión de discos rígidos. En la versión que ahora nos ocupa, la 1.0, el ordenador almacena todos los ficheros de datos bajo un directorio común. Ello equivale al hecho de guardar a la totalidad de los datos dentro de carpetas alojadas en un mismo cajón, sin disponer de otro elemento identificador que su nombre a la hora de proceder a cualquier operación de búsqueda.

La esencia de este sistema operativo reside, tal como se ha indicado, en la gestión de los archivos residentes en disco flexible. Las unidades de disco asociadas al equipo se identifican, de forma consecutiva, por medio de las letras A, B, C... En el caso de la versión DOS 2.0, las unidades de disco rígido adoptan un criterio de identificación que se describirá en un capítulo posterior.

Debido a su propia estructura, este sistema operativo necesita la presencia de dos unidades de disco. Si sólo dispone de una unidad, el propio MS/DOS se encargará de simular la presencia virtual de la segunda unidad de disco. Ello no afectará al modo habitual de operación, si bien, el usuario se verá obligado a cambiar continuamente de disquete, con la consiguiente deceleración en la velocidad de trabajo.

De cara a la búsqueda de información en disquete, el sistema operativo debe conocer cuál es la unidad de disco en la que reside. Ello se indicará suministrando al DOS la clave de la unidad implicada. De omitir tal indicación, el sistema operativo toma por defecto una de ellas para buscar la información solicitada.

La referencia a la unidad implicada aparecerá en el "prompt" o indicador de presencia, quien revela que el ordenador está en disposición de recibir una orden:

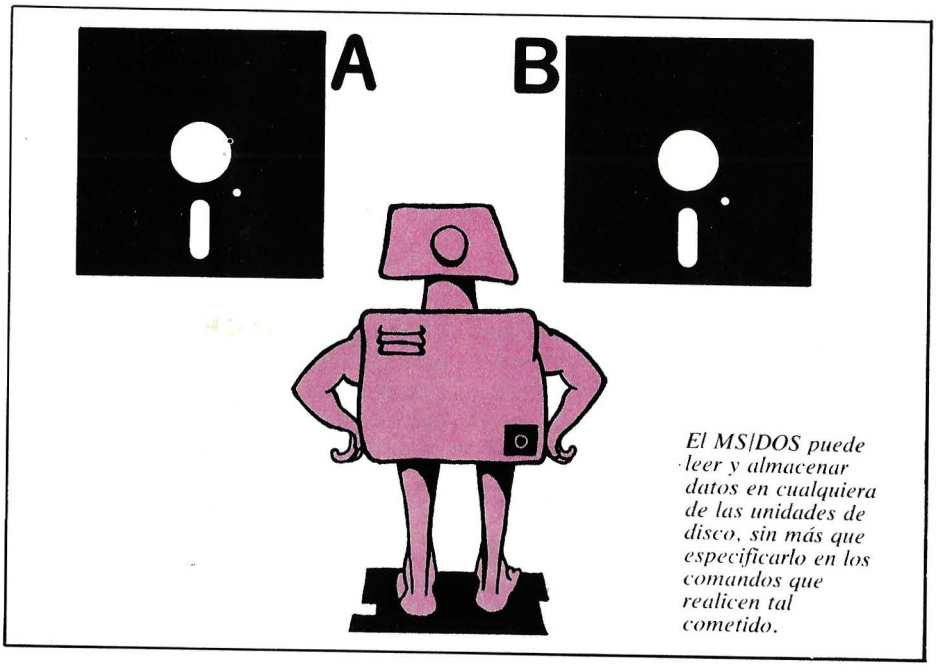
A>Unidad de disco flexible A.
B>Unidad de disco flexible B.

Si se desea cambiar la unidad de disco tomada como opción por defecto, por ejemplo de A a B, bastará con comunicarlo al ordenador como sigue:

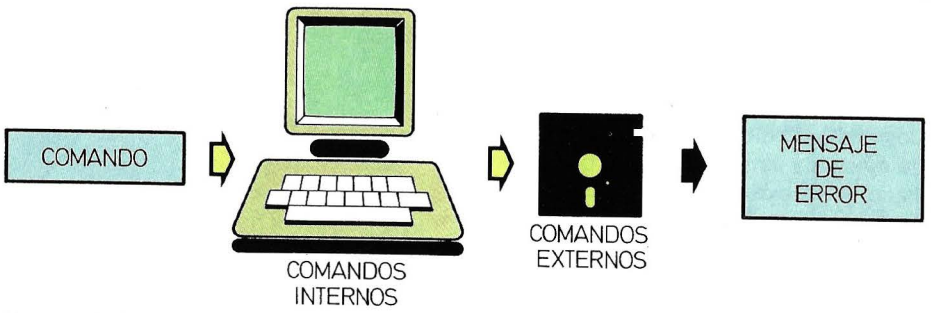
A>B: <CR>

(CR=retroceso de carro; tecla RETURN o ENTER)

A partir de este preciso instante, la unidad de disco B será la que adopte el sistema por defecto. Su indicativo (B) pasará a sustituir a la letra A en el "prompt".



El MS/DOS puede leer y almacenar datos en cualquiera de las unidades de disco, sin más que especificarlo en los comandos que realicen tal cometido.



Proceso de búsqueda de un comando por parte del sistema operativo MS/DOS.

TIPOS DE COMANDOS

Para que el ordenador sea capaz de acometer cualquier tarea, es preciso que reciba las órdenes adecuadas en forma de comandos.
En esencia, los comandos del DOS son, realmente, programas activados por la introducción de su respectivo nombre. Como tales programas, cabe preguntarse cómo se almacenan y cómo son ejecutados por la máquina, teniendo en cuenta su disponibilidad ante los requerimientos del usuario.
Al igual que sucede con el sistema operativo CP/M, el MS/DOS incorpora toda una serie de comandos que residen habitualmente en la memoria central del equipo. Estos pasan a la memoria interna durante el proceso inicial de carga del sistema operativo. Dada su localización durante los períodos de funcionamiento del ordenador, estos comandos se denominan *internos*.
En este aspecto, el MS/DOS es superior al CP/M: posee un mayor número de comandos internos y éstos poseen un formato más lógico.
En todo caso, el número de comandos internos no puede crecer en exceso, dado que el ordenador tiene un limitado volumen de memoria primaria. Resulta obvio que cuantos más comandos residan en su interior, tanto menor será el espacio de memoria que quedará a disposición de los programas de usuario. En virtud de esta limitación, se ha llegado a un compromiso: una serie de comandos, denominados externos, permanecen almacenados en el disquete que contiene el sistema operativo, concretamente en ficheros de tipo COM. Cada vez que es necesaria su presencia, éstos son cargados en la memoria interna como paso previo a su ejecución.
Bajo tal perspectiva, cuando se invoca a un comando MS/DOS, pueden suceder dos cosas. Si el comando es interno, éste se ejecutará inmediatamente; por el contrario, si es externo, el sistema operativo no lo encontrará en su lista de comandos internos y, en consecuencia, procederá a buscarlo en el disquete que se encuentre en la unidad considerada por defecto. Este aparecerá en forma de fichero de tipo COM. Una vez localizado, lo trasladará a la memoria central y procederá a ejecutarlo. Desde luego, todo este proceso supone el consiguiente retardo.
Al realizar semejante operación, el ordenador puede presentar en la pantalla un mensaje de error; mensaje que puede obedecer a dos motivos:

— Se ha escrito de forma incorrecta el nombre del comando.

— Aun siendo correcta su denominación, se trata de un comando externo cuyo respectivo fichero no se encuentra almacenado en el disco en el que se produce la búsqueda.

NOTACION DE LOS COMANDOS DEL DOS

El MS/DOS es un sistema operativo orientado principalmente al ordenador (como contrapartida a los sistemas operativos orientados al usuario). De ahí que exija al usuario el aprendizaje de los distintos comandos utilizables, así como de su formato.
El formato genérico de los comandos del MS/DOS adopta la siguiente disposición:

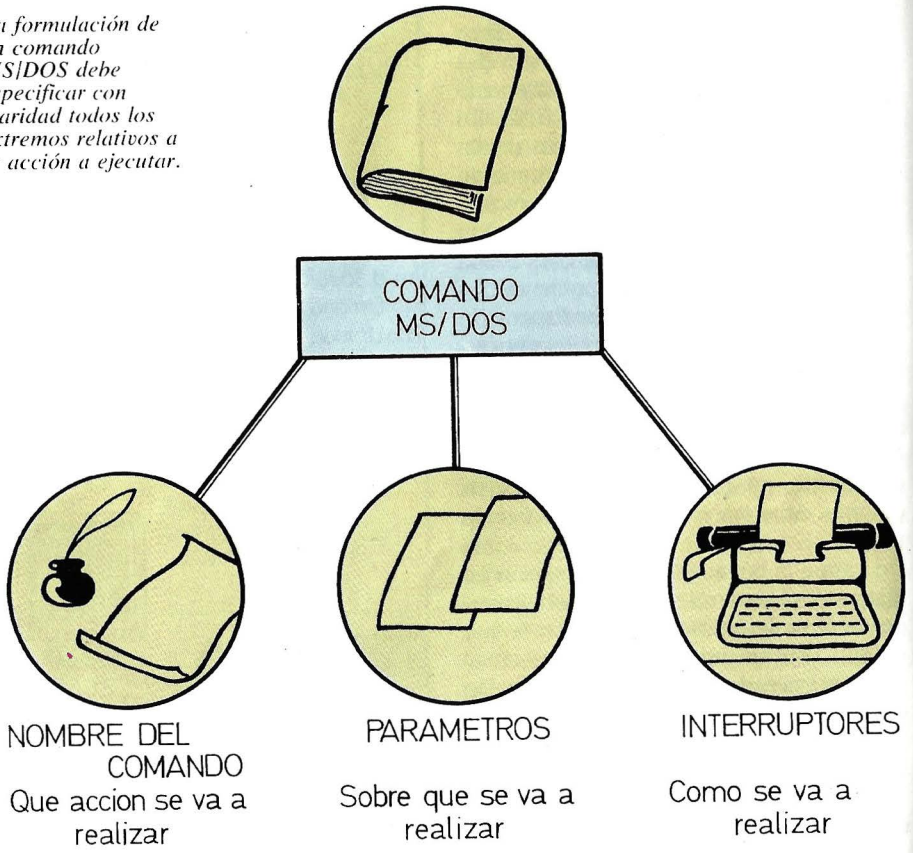
>NOMBRE DEL COMANDO PARAMETROS

Cada una de ambas zonas realiza una función dentro de la orden.
— Nombre del comando: indica la acción a realizar.
— Parámetros: definen sobre qué va a actuar el comando y cómo va a realizar dicha tarea. Los parámetros que se ocupan de este segundo cometido se denominan interruptores (switches); normalmente coinciden con los símbolos "/" o "–", dependiendo del equipo.
Un ejemplo práctico lo constituyen los siguientes comandos, adecuados para copiar archivos:

> COPY SOFT .TEX RESUMEN .TEX
> COPY A:*.* B:

El primero de ellos copiará el fichero SOFT.TEX en otro cuya denominación es RESUMEN.TEX. Al no indicarse el disco afectado, se da por supuesto que ambos

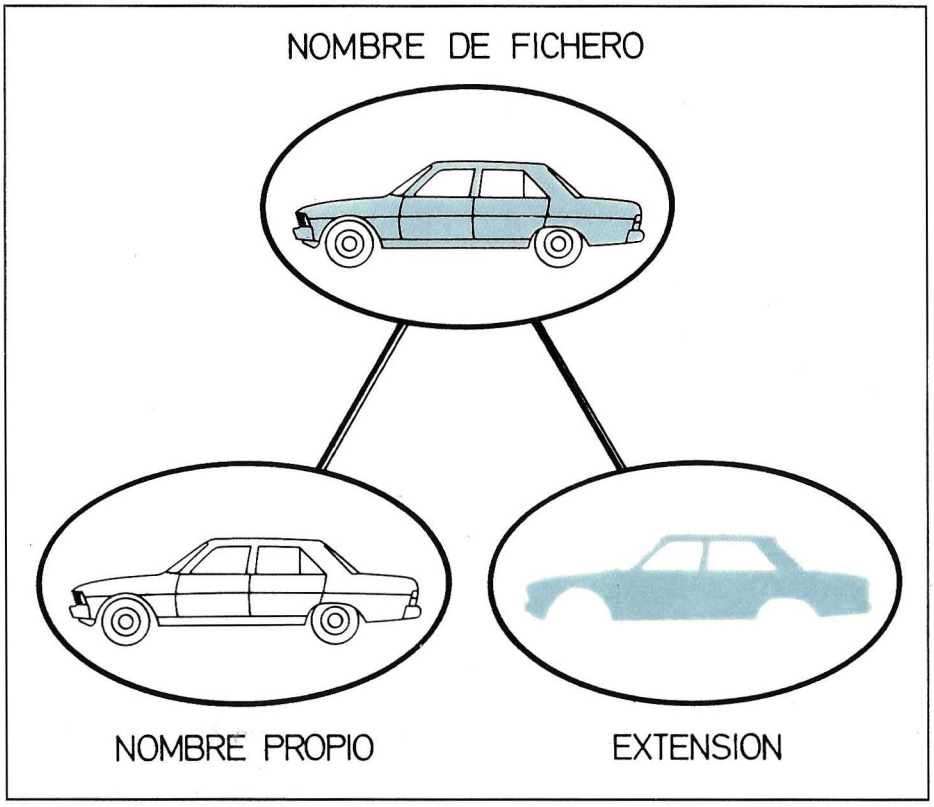
La formulación de un comando MS/DOS debe especificar con claridad todos los extremos relativos a la acción a ejecutar.



ficheros pertenecen al disco insertado en la unidad que el sistema toma por defecto en ese preciso instante. El indicativo de la mencionada unidad aparecerá en el "prompt", delante del símbolo ">".
El segundo comando incluye de nuevo dos parámetros: uno para especificar cuál es la información a copiar (todos los archivos del disco A) y otro para indicar cuál es el destino de la copia (el disco B). Como respuesta, el ordenador copiará todos los ficheros del disco A en el disco situado en la unidad B, otorgando a los ficheros de destino el mismo nombre que poseen sus respectivos ficheros originales.
La sintaxis de este segundo comando presenta algunas particularidades que posteriormente se analizarán con mayor detalle. Tal es el caso, por ejemplo, del uso de los asteriscos al definir los ficheros a copiar. Por el momento es suficiente con saber que el parámetro "A:*.*", alude a todos los ficheros almacenados en el disco A.
En ambos ejemplos, la separación entre los distintos comandos coincide con un espacio en blanco. Algunos equipos admiten otros caracteres, además del espacio en blanco, como separadores de parametros: coma (,), punto (.), punto y coma (;)...

LOS FICHEROS Y SU Y SU DENOMINACION

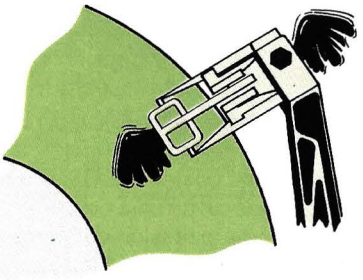
Los programas, datos y comandos externos del sistema operativo MS/DOS se almacenan en disco adoptando la forma de ficheros. Para identificarlos unívocamente, cada uno de ellos recibe un nombre. Este ha de ser forzosamente distinto para cada uno de los ficheros, en versión DOS 1.0.
En la versión 2.0, que se estudiará en un próximo capítulo, es posible encontrar ficheros con el mismo nombre; si bien, el proceso de creación de los mismos es tal que permite diferenciarlos uno de otro.
La identificación completa de un fichero consta de dos zonas, separadas entre sí por un punto. La primera zona es imperativa, mientras que la segunda es opcional. Ambas zonas se denominan, respectivamente, *nombre* propio del fichero ("proper") y *extensión* del nombre del fichero.
El propio o "proper" debe estar compuesto por al menos un carácter, y su



La identificación completa de cada fichero exige la presencia del nombre propio y del apellido o extensión. Este último revela el carácter de los datos contenidos dentro del fichero.

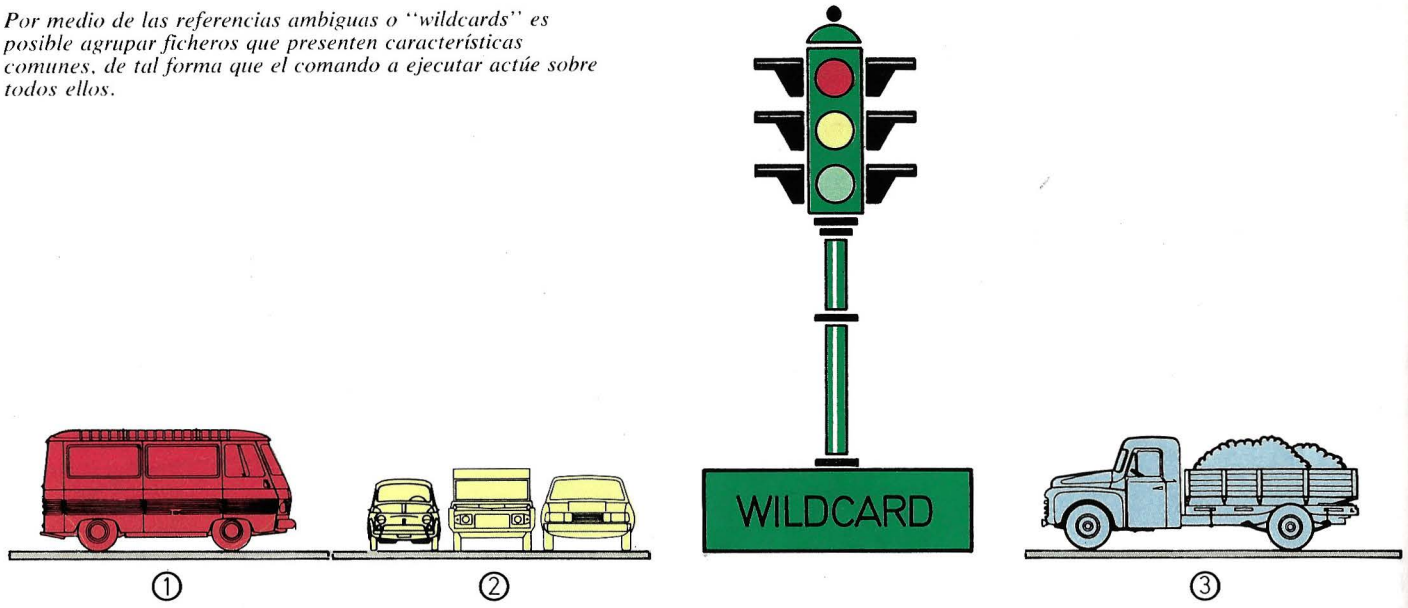
Los discos rígidos

En los últimos años, los discos denominados "rígidos" han alcanzado una gran difusión como método para el almacenamiento permanente de información en los microordenadores. La velocidad de acceso y su elevada capacidad de almacenamiento —de 5 a 40 Mbytes por unidad— supone una sustancial mejora respecto a los discos flexibles. Hasta ahora, el tipo habitual de discos rígidos son de tecnología Winchester. Este apelativo procede del nombre en clave que empleó IBM, la firma creadora de esta tecnología, para denominar el proyecto.



Los discos Winchester son semejantes en su constitución y funcionamiento a los discos de tecnologías anteriores. La mayor divergencia aparece en el método de operación de la cabeza grabadora de datos. El procedimiento, original y altamente eficaz, hace que los discos de esta tecnología resulten más fiables y duraderos. La clave estriba en el hecho de que la cabeza no establece contacto con la superficie del disco, sino que se encuentra separada del mismo a una distancia de medio micrómetro. Para conseguirlo, la cabeza está dotada de tres raíles paralelos. El raíl central es el que se emplea para la grabación y recuperación de los datos, mientras que los laterales gobiernan el flujo de aire. De esta forma, al girar el disco se produce una fuerza aerodinámica que separa la cabeza de la superficie magnética.
El conjunto se almacena en el interior de una caja hermética, en la que se introduce el aire filtrado, a presión. Este se encarga de eliminar las partículas de polvo que podrían interferir en el proceso de lectura y escritura de datos.

Por medio de las referencias ambiguas o "wildcards" es posible agrupar ficheros que presenten características comunes, de tal forma que el comando a ejecutar actúe sobre todos ellos.



longitud máxima puede elevarse a 8 caracteres. Estos pueden ser letras, números o determinados signos de puntuación que no se utilicen en la formulación de comandos. De ahí que no sea admisible la inclusión de puntos ni espacios en blanco dentro del nombre. Algunos nombres correctos de ficheros son los siguientes:
DATOS,A7,BXY23 ó ABC-34
La extensión del nombre del fichero es un apéndice que consta de un máximo de tres caracteres. Su mayor utilidad estriba en que puede dar una idea del tipo de información que contiene el fichero. Por ejemplo, los comandos externos del MS/DOS residen en ficheros cuya extensión es "COM" (extensión indicativa de comando).
Cabe encontrar también una categoría de ficheros, denominados de tipo batch, en los que el usuario almacena un conjunto de comandos del DOS. Los referidos comandos se ejecutarán secuencialmente al invocar al fichero en cuestión, de igual forma que si el usuario los fuera introduciendo sucesivamente a través del teclado. Este tipo de ficheros, cuyo efecto práctico es similar al de los ficheros que acogen a los comandos externos, incluyen como extensión del nombre propio la expresión "BAT".
Un dato a señalar, relativo a la denominación que identifica a los ficheros, es la posibilidad de utilizar *referencias ambiguas* o "wildcards". Su interés estriba en que permiten extender la actuación de un comando a un grupo genérico de ficheros,

Normalmente, los equipos que incorporan el sistema operativo MS/DOS suelen disponer de una doble unidad de disco. En la fotografía aparece el Ericsson PC, un ordenador personal compatible con el IBM-PC y equipado con una doble unidad para discos flexibles.

sin necesidad de especificar el nombre de cada uno de ellos.
En el sistema operativo DOS, la referencia ambigua adecuada para ocupar la zona de nombre o de extensión del nombre es el símbolo "*" (asterisco). Un ejemplo práctico lo constituye la orden COPY formulada en el apartado precedente. El empleo de "*.*" como sustitutivo de la denominación del fichero, hará que sean copiados todos los ficheros del disco origen. Si la designación del fichero coincide con "*.COM", los archivos afectados por la operación de copia serán todos aquellos cuya extensión sea COM, con independencia de su nombre específico.

PFS File-Report (y 2)

El generador de informes PFS-Report

Tras estudiar las características generales de este sistema para la gestión de ficheros, profundizando en el paquete FILE, llega el momento de detallar las posibilidades que ofrece la zona REPORT. Con ello se completará una visión general de este útil y sencillo paquete de aplicación, capaz de ofrecer soluciones eficientes a distintos problemas y con un soporte informático elemental.

FUNCIONAMIENTO DEL PFS-REPORT

El programa PFS-Report tiene básicamente tres opciones: escribir un informe, crear el formato con que se debe escribir el informe y cambiar las cabeceras de página. Tanto para escribir un informe como para cambiar las cabeceras, hay que recurrir al menú de funciones del PFS-Report; en cambio, para crear formatos es preciso dirigirse al menú de formateo.

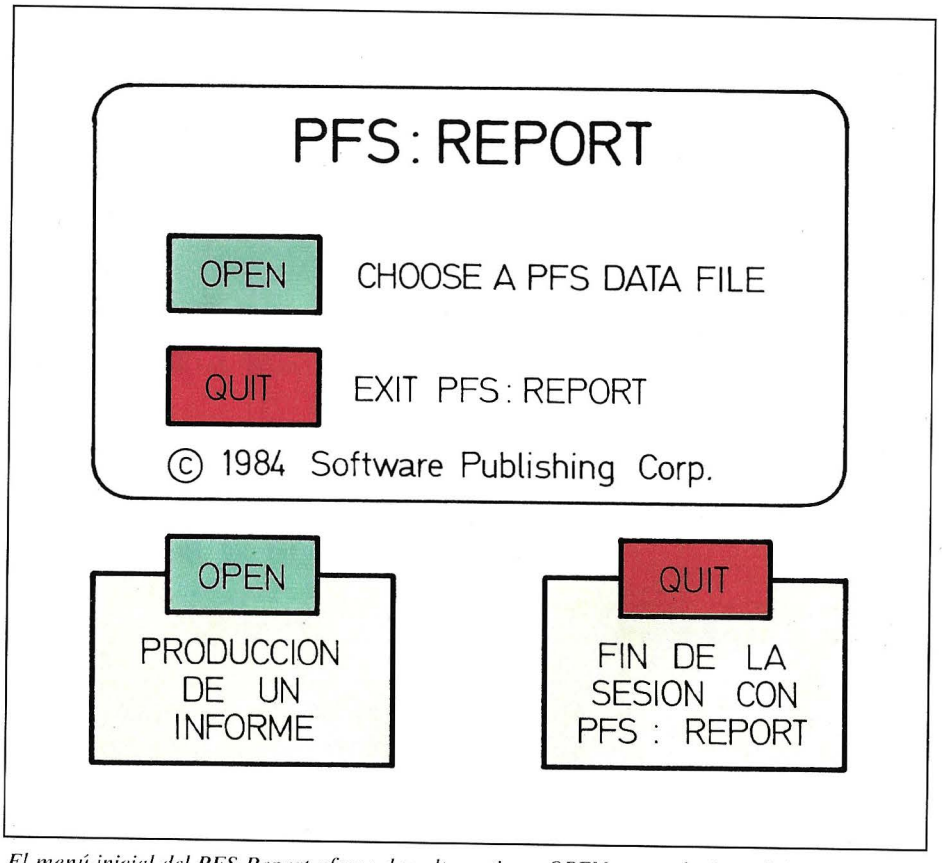
Los ficheros a tratar por el PFS-Report tienen que haber sido creados mediante el programa PFS-File.

Para identificar a cada fichero hay que hacer mención de su nombre; tras ello, se elegirá la opción de trabajo deseada y, por último, habrá que precisar las características de la impresora. Esta plasmará los informes cuya estructura define el usuario apoyándose en las funciones que brinda el paquete PFS-Report. Por supuesto, el diseño del formato de impresión sólo debe realizarse una vez; para los restantes informes que se deseen producir con el mismo formato, bastará con apelar al ya definido. En efecto, es posible almacenar en disco los formatos de impresión otorgándoles un nombre propio. Tras ello la tarea se reducirá a combinarlos con los ficheros de datos.

El paquete contempla la impresión de cabeceras "por defecto". De no especificar lo contrario, se imprimirán en el informe los nombres que están asociados en el fichero a cada dato escrito. No obstante, si se desea modificar esta primera línea de escritura (la cabecera), puede utilizarse la función apropiada para alterarlos a voluntad del usuario. Esta opción es fundamental cuando algún dato tenga reservado un gran espacio en el informe y su nombre o título sea de reducida dimensión. En tal caso, la cabecera aparecería desplazada con el nombre inicial. Una vez invocado el programa PFS-Report, aparecerá un menú en la pantalla aportando dos alternativas:

1. OPEN (Choose a PFS data File: *seleccione un fichero de datos PFS)*

Tras seleccionar esta opción aparecerá un nuevo menú que mostrará los diversos ficheros almacenados en el disco. A continuación, el programa solicitará el nombre del fichero que debe ser procesado y pre-



El menú inicial del PFS-Report ofrece dos alternativas: OPEN, para elegir un fichero y producir un informe acerca del mismo; y QUIT, para dar por terminada la sesión de trabajo.

sentará un nuevo menú que visualiza el nombre del fichero seleccionado y el siguiente mensaje: "Please make a menu selection". Durante toda la sesión con el PFS-Report, está presente este "pantallazo" en el que irán apareciendo los mensajes del programa y las entradas del usuario.

2. QUIT (Exit PFS-Report: *salida del PFS-Report)*
Esta opción sirve para finalizar la sesión de trabajo con el programa PFS-Report. Al seleccionarla, el ordenador queda disponible para que el usuario pueda ejecutar otros programas.

MENUS DEL PFS-REPORT

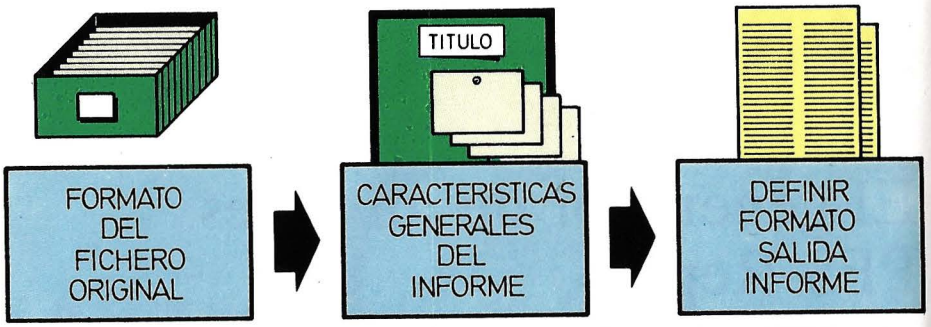
Existen cuatro tipos de menús utilizables desde el PFS-Report: File, Edit, Functions y Formats. Ocasionalmente, según el tipo de operación que estemos realizando, puede aparecer un nuevo tipo de menú denominado Help. Para elegir cuál de ellos se desea utilizar, basta con posicionar el cursor encima del menú elegido y pulsar "ENTER" en el teclado (o realizar la selección directamente por medio del "ratón", en el caso del Apple MacIntosh). Una vez realizada esta operación, aparecerán en la pantalla distintas opciones del menú. La utilidad de cada una de ellas es la siguiente:

1. **FILE**
Se utiliza para gestionar la entrada y salida de la aplicación, así como para llamar a los archivos que aportarán los datos base para emitir los informes.

2. **EDIT**
Ofrece las funciones necesarias para facilitar el trabajo de edición.

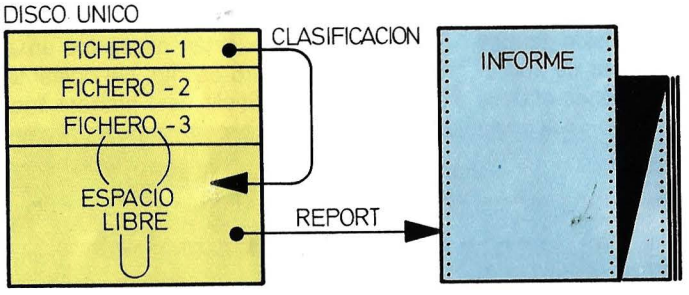
3. **FUNCTIONS**
Permite seleccionar alguna de las funciones que ofrece el PFS-Report para producir informes.

4. **FORMATS**
Se utiliza para establecer los formatos de escritura de los informes.

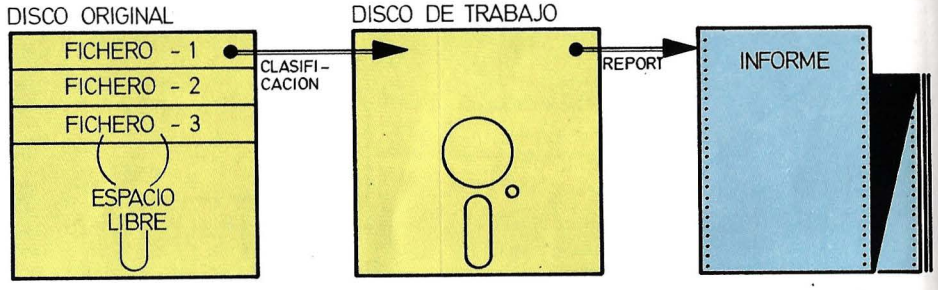


La figura ilustra los tres pasos necesarios para producir un informe mediante el PFS-Report: describir el formato del fichero original, indicar las características generales del informe y definir el formato de salida.

5. **HELP**
Tan sólo cuando se realizan operaciones de formateo de informes, es posible utilizar el menú de ayuda HELP. Sirve para que el usuario pueda cerciorarse de los posibles errores que esté cometiendo.



Cuando el PFS-Report necesita clasificar un fichero y el equipo no dispone más que de una unidad de disco, se corre el riesgo de que el espacio disponible no sea suficiente para realizar la clasificación...



... En cambio, si se utiliza una segunda unidad de disco para el fichero de trabajo, quedará garantizada la posibilidad de realizar la clasificación.

PRODUCCION DE UN INFORME

En este apartado se detallan los distintos pasos que conducen a la obtención de un informe a partir de un fichero PFS. Para ello, es necesario utilizar tres menús. El primero para indicar el formato con que están almacenados los datos en el fichero; éste coincide con el diseño que se realizó mediante el programa PFS-File. El segundo para definir las características generales del informe: título general, tamaño de cada página... Y el tercero para indicar qué datos deben aparecer en el listado, en qué columnas se ubicarán y en qué orden se producirá el informe.
Suponga que el punto de partida es un fichero PFS, almacenado en disco, cuyo nombre es PERSONAL. Su contenido son datos relativos a los empleados de una empresa. A partir del mismo se desea producir un simple listado con los nombres y direcciones de todos los empleados de la empresa.
La primera operación a realizar no es otra que "arrancar" el programa PFS-Report, de la forma descrita en el apartado de

funcionamiento: se elegirá la opción OPEN y se indicará "PERSONAL" como nombre del fichero a partir del que se desea producir el informe.
Acto seguido, el programa solicitará al usuario que decida el menú que desea utilizar. Este optará por el menú de funciones y, dentro de él, seleccionará la función "PRINT A REPORT" (escribir un informe). Inmediatamente, aparecerá en la pantalla el formulario propio del fichero en cuestión (figura adjunta).
El siguiente paso consiste en seleccionar los registros del fichero que se van a incluir en el informe. Si el listado debe incluir a todos los registros —tal es el caso de nuestro ejemplo—, no será preciso realizar operación alguna. No obstante, si el usuario desea introducir alguna condición de selección, se verá obligado a invocar la función "FIND FORMS", de forma análoga a la ya descrita en el apartado "Inclusión y extracción de datos del fichero" (ver capítulo anterior).
Una vez introducidas las condiciones, se seleccionará la opción "PRINT REPORT" (impresión del informe).
Antes de imprimir físicamente el informe, el programa es capaz de mostrar en pantalla el número de registros que han superado las restricciones impuestas. En el ejemplo propuesto, todos los registros satisfarán las condiciones de impresión y, en consecuencia, este dato coincidirá con el número total de empleados de la empresa. El próximo menú que ocupará la pantalla, permite definir las características formales del informe: el número de líneas por página, título del informe... La estructura de dicho menú es análoga a la que corresponde al PFS-File.
El último paso necesario para disponer del informe escrito, consiste en especificar el formato de salida, esto es: qué datos van a salir, en qué columnas, y cuál será el orden de los registros que superen las restricciones.
Cada informe puede incluir hasta un máximo de 20 columnas de datos, columnas que se numerarán correlativamente desde la 1 hasta el valor coincidente con el

Bases de datos jerárquicas

Para llegar a definir con propiedad el concepto de base de datos jerárquica, hay que precisar antes el significado de dos conceptos en el entorno de este modelo:

- **SEGMENTO**

Consiste en un conjunto de datos de un determinado tamaño, que contiene información de entidades homogéneas.

- **REGISTRO**

Conjunto de segmentos jerarquizados, de tal forma que la relación entre ellos coincide con alguna de las siguientes:

1. *Segmento padre.*
Se dice que un segmento A es padre de otro B, cuando B desciende directamente de A.

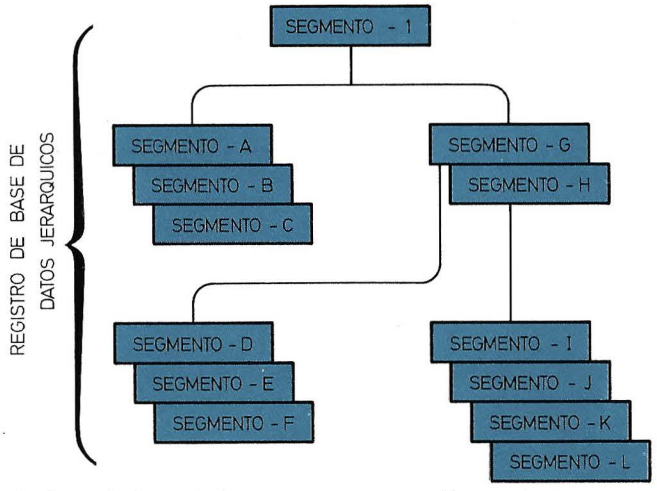
2. *Segmento hijo.*
Un segmento A es hijo de otro B, cuando A desciende directamente de B.

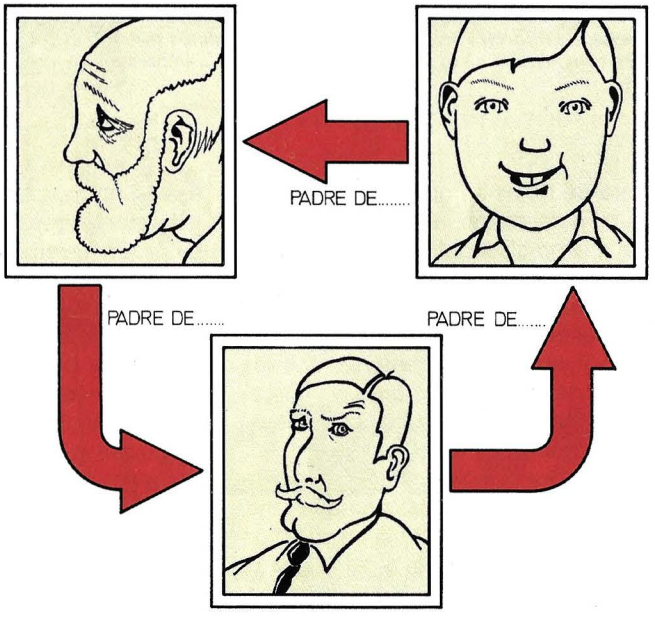
3. *Segmentos hermanos.*
Los segmentos A1, A2, ... An son hermanos, si todos ellos descienden de un mismo padre.

4. *Segmentos mellizos.*
Los segmentos A1, A2, ... An son mellizos, si son hermanos y además del mismo tipo.

5. *Hoja.*
Un segmento A es una hoja si no tiene ningún descendiente.

A partir de estas consideraciones previas, puede definirse a la base de datos jerárquica, de forma inmediata: "una base de datos jerárquica es un conjunto ordenado de registros del tipo enunciado".
Probablemente sea éste el modelo de base de datos más extendido en aplicaciones empresariales de tipo corporativo. Permite establecer un conjunto cerrado de relaciones entre los datos (padre, hijo, hermano, ...); no obstante —y aquí reside su principal defecto—, este modelo no permite plasmar relaciones extrañas: por ejemplo, que el hijo de un segmento sea el padre de su padre.





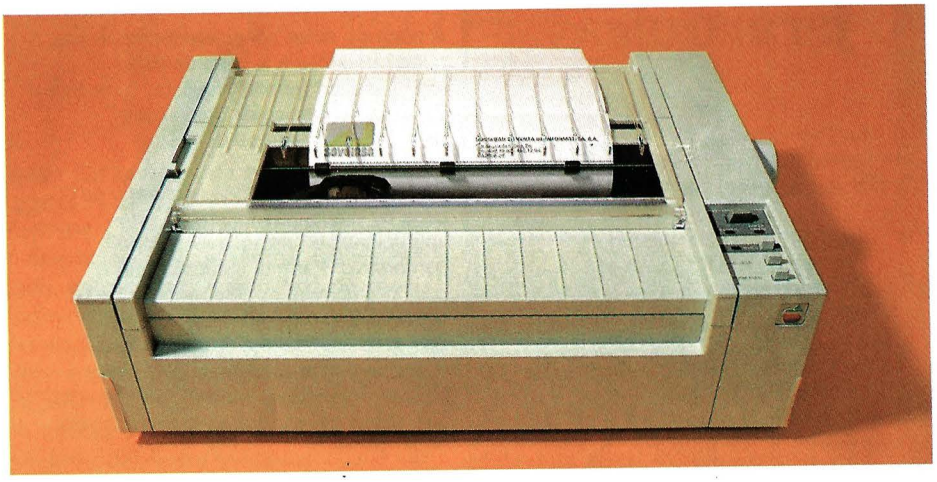
La base de datos jerárquica no permite establecer relaciones extrañas; por ejemplo, que el hijo de un segmento sea, a su vez, el padre de su segmento padre.

```
EMPLEADOS
Page 1 -- please choose forms for the report.        Print report

Numero de empleado:
-------------------------------------------------------------------

Nombre del empleado:1
Antiguedad:
Categoria profesional:
Edad:                                  Sexo:
Direccion:2
Localidad:

-------------------------------------------------------------------
```

En la pantalla reproducida en la figura, se solicita un informe del fichero cuyo nombre es "PERSONAL". En el informe sólo aparecerán los nombres (1) y las direcciones de los empleados (2).

```
PERSONAL
9 forms printed -- please make a menu selection.

                    RELACION DE EMPLEADOS

Nombre del empleado          Direccion
--------------------         ----------
Abad Lopez, Pedro            Pza. Juan Tello, 45
Benitez Palacios, Juan       C/ La Albufera, 56
Giner Duarte, Mariano        C/ Lagasca, 45
Gomez Rodriguez, Jose        Pza. Olavide, 22
Marino Diaz, Benito          C/ Orense, 11
Perez Nieto, Vicente         C/ Sagasta, 128
Sanchez Agesta, Manuel       C/ Rios Rosas, 32
Verdu Jimenez, Rosario       C/ Pez, 5
```

Informe obtenido a partir de la selección realizada en el menú precedente. Este incluirá a todos los empleados clasificados por orden alfabético.



El PFS-Report es un paquete de aplicación complementario del PFS-File. Su especialidad es la obtención de informes selectivos a partir de los datos tratados con el PFS-File. Los informes pueden proyectarse en la pantalla o volcarse en la impresora conectada al equipo.

número de datos a imprimir. Para proceder a tal numeración, aparecerá un menú con el mismo diseño del fichero; en él se especificarán los números de orden, dejando en blanco los campos de datos que no se deseen imprimir. Por lo tanto, en nuestro ejemplo, sólo numeraremos los campos de datos "nombre" y "dirección", ya que son los únicos que deseamos que aparezcan en el informe.

El orden de numeración no sólo sirve para asociar datos a columnas, sino que también define implícitamente la clasificación del informe. Por ejemplo, si se asigna el número 1 al nombre y el 2 a la dirección, el listado aparecerá clasificado alfabéticamente por nombres empleados. Si dos o más empleados tienen el mismo nombre, el orden quedará determinado por el campo de dirección. De especificar el orden inverso —esto es: dirección 1 y nombre 2—, la clasificación tendrá lugar en función de las direcciones, y dentro de ellas por nombres.

Al concluir todos los pasos enunciados, empezará la impresión del informe. Si por cualquier motivo se desea interrumpir la confección del listado, basta con utilizar el comando adecuado para cancelarla; de no ser así, el programa continuará escribiendo hasta terminar con toda la información del fichero.

RESTRICCIONES EN LA CLASIFICACION DE DATOS

El PFS-Report puede clasificar los registros con sólo especificar el usuario qué datos son los que hay que considerar al efecto. Para realizar tal clasificación, el programa necesita utilizar un fichero de trabajo, en el que irá modificando el orden inicial del fichero de origen. De esta forma, al terminar el proceso de clasificación, el fichero inicial se mantendrá sin modificaciones, mientras que el fichero de trabajo memorizará una copia del original convenientemente ordenado.

Si el ordenador sobre el que se ejecuta el paquete PFS-Report dispone de una segunda unidad de disco, ésta se utilizará para almacenar al fichero de trabajo. En caso contrario, habrá que trabajar con ciertas limitaciones.

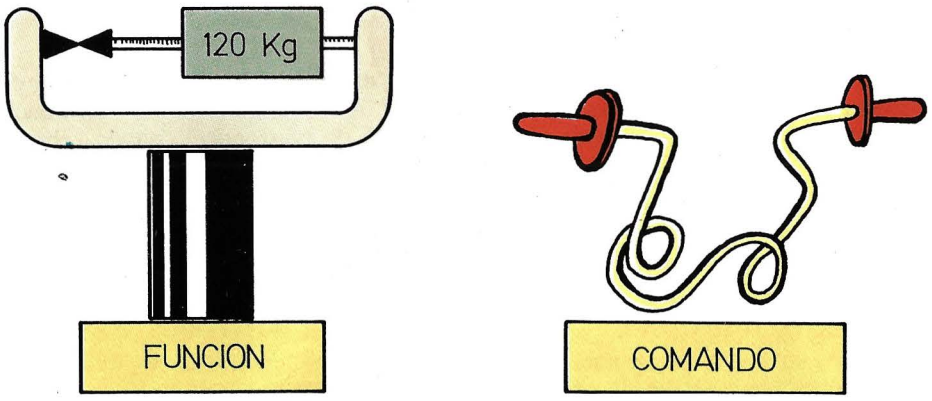
Dependiendo del espacio que quede libre en el propio disco que contiene al fichero PFS, será posible o no realizar la clasificación. De no existir suficiente espacio libre, el programa avisará al usuario mediante el mensaje "The disk is full" (el disco está lleno). A partir de ahí, para que sea factible obtener el informe, el usuario se verá obligado a modificar las especificaciones de clasificación. Si en tal circunstancia no se desea modificar la estructura del informe, habrá que ordenar la copia y posterior borrado de alguno de los ficheros residentes en el disco, con objeto de liberar el espacio suficiente para la clasificación.

Funciones BASIC

Introducción de datos, funciones numéricas y de azar

La mayor parte de las palabras BASIC mencionadas hasta ahora son comandos. En el primer capítulo dedicado al manejo de cadenas alfanuméricas se introdujeron palabras que no eran comandos, sino funciones. La diferencia entre ambos tipos de órdenes estriba en su funcionamiento: si los comandos realizan acciones, las funciones ejecutan operaciones. Ello significa que el efecto de una función será emitir un resultado en forma de dato.

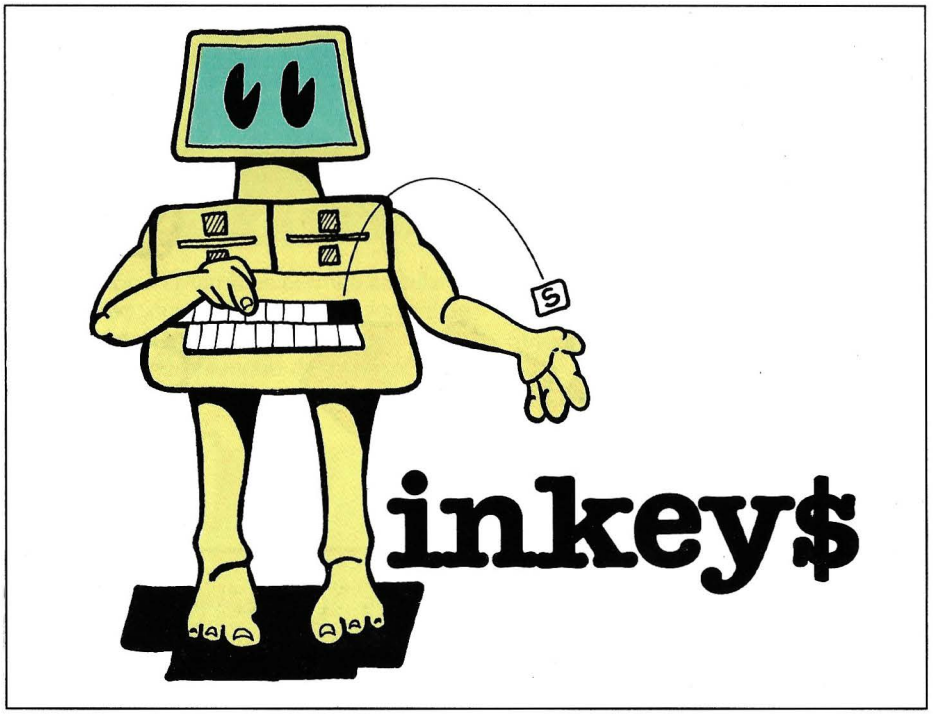
El uso del comando PRINT dentro de una instrucción, por ejemplo, PRINT "HOLA", produce una acción; no emite un dato resultante de una operación. Dicha acción corresponde con la presentación en pantalla de la palabra HOLA. Sin embargo, la función LEFT$ (1, "HOLA") ejecuta una operación cuyo resultado es el dato "H". Así pues, función es toda herramienta del vocabulario BASIC que proporciona un dato. A lo largo del presente capítulo se van a estudiar algunas funciones de interés, que servirán para sacar un mayor partido a la máquina.



La diferencia entre una función y un comando reside en el hecho de que la función proporciona un dato de salida, mientras que el comando realiza una acción.

INTRODUCCION DE DATOS

Cuando en un programa es necesario introducir nuevos datos en cada ejecución, ello debe indicarse dentro del propio programa. Tal cometido se realizaba hasta ahora por medio del comando INPUT. A pesar de su indudable utilidad, éste comando presenta ciertos inconvenientes. Cuando la cantidad de datos a introducir es grande y se exige cierta rapidez, la actuación de INPUT deja bastante que desear, puesto que obliga a pulsar la tecla RETURN tras cada dato introducido. El BASIC incorpora una función especializada en la introducción de datos. Esta es INKEY$, la cual proporciona el carácter pulsado en el teclado. La diferencia entre



INKEY$

Recoge un carácter pulsado en el teclado.

Formato: <var.>=INKEY$

Ejemplos: 70 K$=INKEY$
50 B$=INKEY$

INPUT e INKEY$ se observará con claridad a través del siguiente ejemplo.
A menudo, se utilizan decisiones aportadas por el usuario para desviar la ejecución del programa. Apelando al comando INPUT, la forma de resolver tal necesidad puede coincidir con la que refleja el siguiente bloque de instrucciones:

```
...
200 PRINT "DESEA CONTINUAR (S/N)?"
210 INPUT K$
220 IF K$="S" THEN GOTO ...
230 IF K$="N" THEN GOTO ...
...
```

Al llegar a esta zona del programa, el operador ha de teclear un dato (S ó N) y luego pulsar RETURN.
Si opta por utilizar la función INKEY$, ya no será necesario accionar la tecla RETURN, puesto que dicha función recoge un solo carácter del teclado. Sin embargo, INKEY$ no detiene la ejecución, con lo cual, si no se pulsa la tecla en el momento oportuno, no se obtendrá nigún dato. En consecuencia, no será suficiente con cambiar la línea 210 por:

```
210 K$=INKEY$
```

Para evitar que la ejecución "pase de largo" es necesario crear un bucle; bucle que ha de permanecer cerrado sobre sí mismo hasta que se detecte la pulsación. Las instrucciones necesarias para programarlo se pueden agrupar en una misma línea:

```
210 K$=INKEY$: IF K$="" THEN GOTO 210
```

Una vez que esta línea entre en actividad, la secuencia de ejecución permanecerá virtualmente bloqueada en la misma hasta que se accione una tecla.
En definitiva, el nuevo aspecto del ejemplo resuelto con la función INKEY$ será el que sigue:

```
...
200 PRINT "DESEA CONTINUAR (S/N)?"
210 K$=INKEY$: IF K$="" THEN GOTO 210
220 IF K$="S" THEN GOTO ...
230 IF K$="N" THEN GOTO ...
...
```

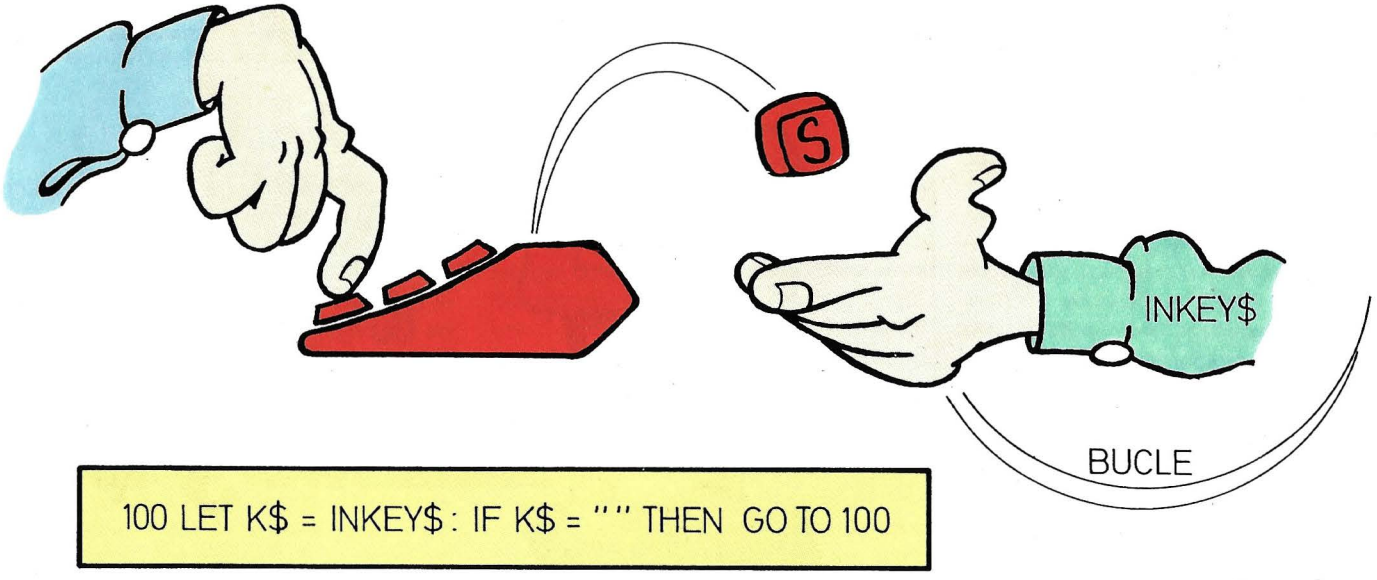
En algunos ordenadores no existe la función INKEY$. Cuando esto ocurre, suele utilizarse en su lugar un nuevo comando: GET. Lo mismo que antes se conseguía con LET K$=INKEY$, se obtiene ahora con GET K$. Aplicando el nuevo comando al ejemplo anterior, sólo habría que cambiar una línea:

```
210 GET K$:IF K$="" THEN GOTO 210
```

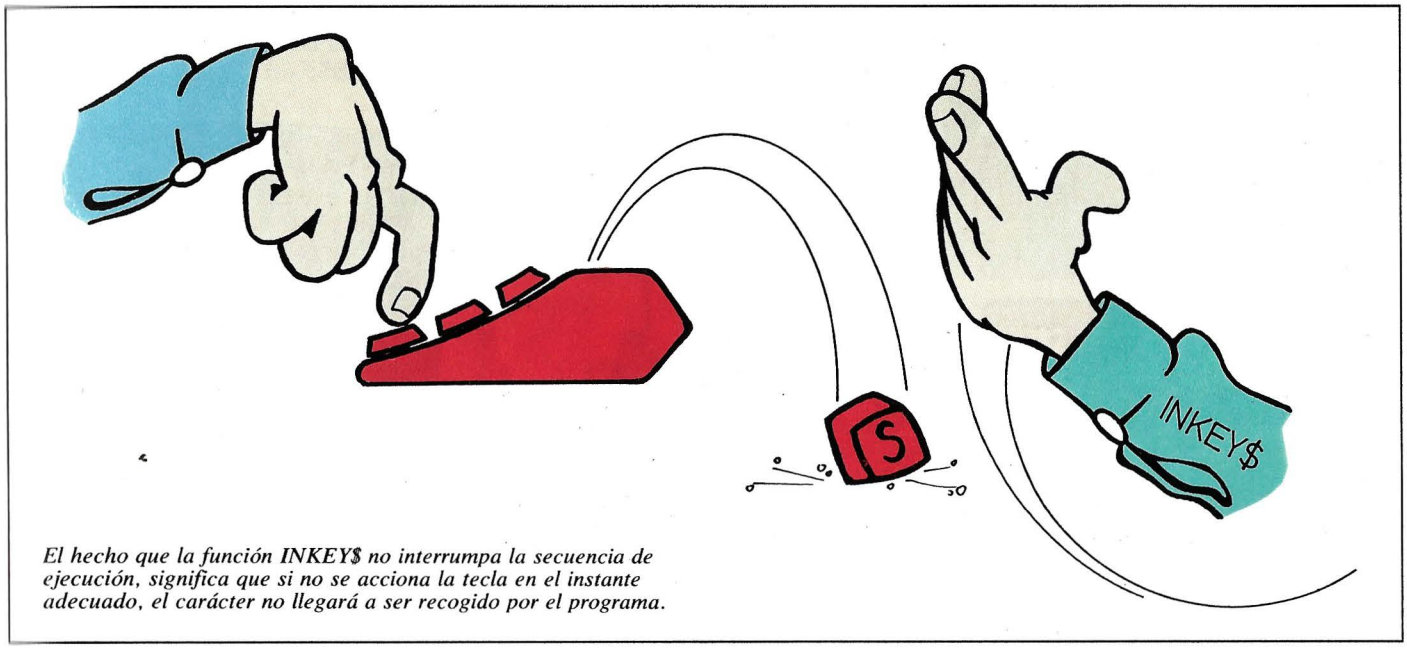
Los bucles creados con este procedimiento, detienen la ejecución del programa hasta que se pulsa una tecla. Ello puede servir para mantener un texto o un mensaje en la pantalla el tiempo que el usuario desee. Cuando se accione una tecla, la ejecución proseguirá. Por ejemplo:

```
...
1000 REM CARTEL
...
2000 PRINT "PULSE UNA TECLA PARA
     SEGUIR"
2010 LET K$=INKEY$: IF K$="" THEN
     GOTO 2010
2020 LET K$=""
...
```

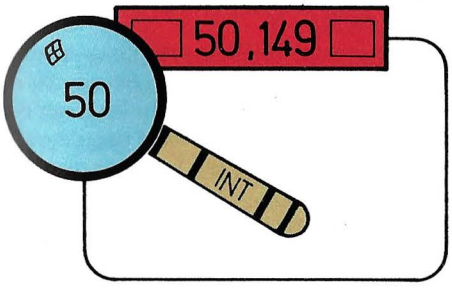
El cartel creado entre las líneas 1000 y 2000 permanecerá en la pantalla mientras se ejecute el bucle. Al pulsar una tecla, K$ deja de ser nulo y, por lo tanto, se produ-



La función INKEY$ recoge el carácter introducido a través del teclado. Como quiera que su presencia no detiene la ejecución del programa, es necesario construir un bucle de espera para la captación del carácter.



El hecho que la función INKEY$ no interrumpa la secuencia de ejecución, significa que si no se acciona la tecla en el instante adecuado, el carácter no llegará a ser recogido por el programa.



El cometido de la función INT es extraer la parte entera del dato numérico que se indique en el argumento.

cirá la salida del bucle. La línea 2020 "borra" el carácter K$, ya que no tiene importancia alguna el valor de la tecla pulsada.

JUGANDO CON NUMEROS

Suponga un programa en el que se desea realizar dos acciones distintas, dependiendo de si un número es par o impar. La forma de resolverlo es haciendo uso de la instrucción IF/THEN. Pero, ¿cómo detectar si el número es par?
Sabemos que un número es par si es divisible por dos. Así, por ejemplo: 6/2=3 y 5/2=2,5; el número 6 es par mientras que el 5 es impar. La diferencia radica en la parte fraccionaria, de tal forma que un número es par si el resultado de dividirlo por dos no posee parte fraccionaria.
Para separar la parte entera de la parte fraccionaria se dispone, en BASIC, de la función INT. Esta devuelve la parte entera (INTeger, entero) del número situado en su argumento:

```
PRINT INT(16.123)
16
■
```

La función INT también permite calcular la parte fraccionaria. La parte fraccionaria de un número N será N-INT(N); esto es: el mismo número aunque desprovisto de la parte entera. A continuación se evalúa la parte fraccionaria del número del ejemplo anterior:

```
PRINT 16.123-INT(16.123)
0.123
■
```

Volviendo al problema del principio, se utilizará la función INT para detectar cuándo un número es par. Resulta obvio que la parte entera de un número entero coincidirá con el mismo número. Luego si N es entero INT(N) y N tendrán el mismo valor. Esto es, precisamente, lo que se va a utilizar para decidir si el dato NUM es par.

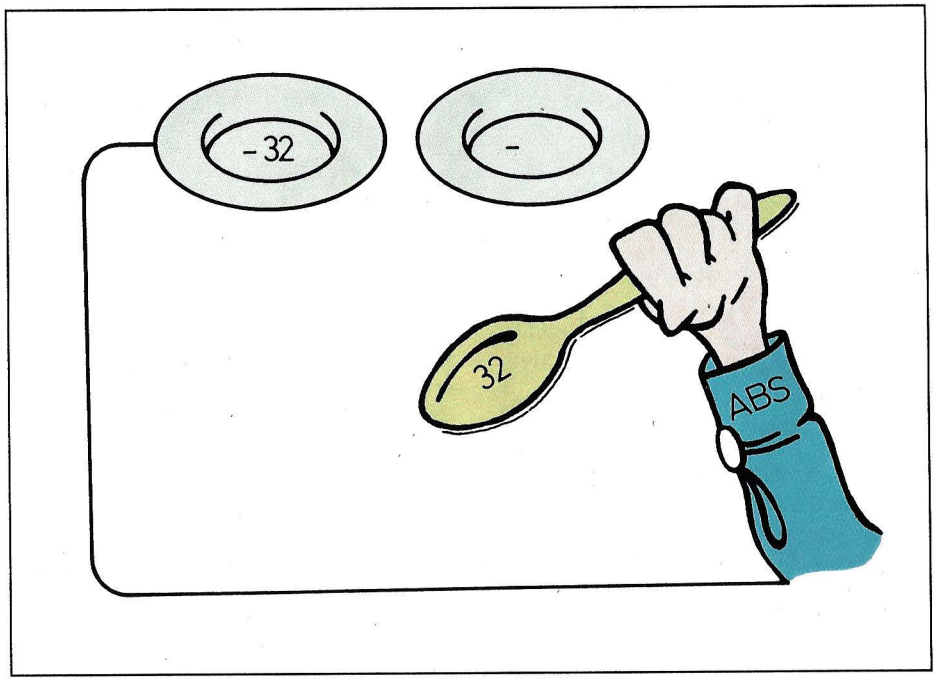
```
IF NUM/2=INT(NUM/2) THEN PRINT "PAR"
```

INT

Devuelve la parte entera de un número.

Formato: *INT(<número>)*

Ejemplos:
```
10 A=INT(17.5)
50 PRINT INT(B+C)
```



Para obtener el valor absoluto de un número, hay que recurrir a la función ABS.

De esta forma, si NUM/2 es igual INT (NUM/2) entonces es que NUM/2 es entero. Y si NUM/2 es entero, resultará que NUM es par.
El mismo procedimiento se puede utilizar para averiguar si NUM es divisible por cualquier otro número. Sencillamente, cambiando el 2 por el número deseado. NUM/7=INT(NUM/7) será cierto si NUM es divisible por siete.
El uso de la parte entera adquiere una importancia capital en determinados casos. Un ejemplo lo constituye el hecho de utilizar PRINT AT X, Y con valores calculados de X e Y. Como ejemplo se propone el diseño de un programa que dibuje un gráfico de barras. Para ello se partirá de un valor máximo de 100, al que corresponderá una altura de 12 filas. La barra se dibujará situando caracteres X en la posición adecuada.

```
100 FOR Y=0 TO DATO STEP 100/12
110 PRINT AT (X,12-INT(Y)); "X"
120 NEXT Y
130 END
```

Aquí es fundamentalmente el uso de la función INT puesto que un número fraccionario de filas no tiene sentido.

MAS FUNCIONES: CUESTION DE SIGNOS

Hay otras funciones que proporcionan resultados interesantes. Una de ellas es ABS, cuyo cometido es calcular el valor absoluto de un número. El siguiente ejemplo es una muestra de las posibilidades de ABS:

```
PRINT ABS(17)
 17
PRINT ABS(-17)
 17
■
```

La función ABS resulta especialmente útil para determinar la separación entre dos números (sean A y B). Si se realiza la resta A-B, el resultado tomará un valor positivo si A es mayor que B. en caso contrario, el resultado será negativo. Para obtener siempre un resultado positivo basta con poner ABS(A-B).
Otra utilidad de ABS se presenta a la hora de calcular raíces cuadradas. Para evitar el error resultante de intentar hallar la raíz de un número negativo, se hará uso de esta función. El método consistirá en escribir SQR(ABS(DATO)), siendo DATO el número al cual se desea extraer la raíz cuadrada.
Una función íntimamente relacionada con ABS es SGN. Como su nombre indica, SGN identifica el "SiGNo" de su argumento. Entrega el valor 1 si el dato es positivo, -1 si es negativo y 0 si el dato que figura en su argumento es nulo.
Una aplicación de SGN puede ser la de adecuar el incremento de un bucle.

```
FOR I=A TO B STEP SGN(B-A)
```

Se supone que el incremento ha de ser de una unidad y los valores de A y B no son conocidos. En consecuencia, si B es mayor que A el incremento ha de ser positivo, mientras que si es A el mayor, el incremento debe ser negativo. Ello se consigue con la cláusula STEP SGN(B-A). Cabe comprobar que cuando B es mayor que A, SGN(B-A) vale 1, y en caso contrario -1. Justamente lo que se pretendía conseguir. Si se desea un incremento de valor distinto de 1, e igual al valor INC, será suficiente con cambiar la parte STEP de la siguiente forma:

```
FOR I=A TO B STEP INC*SGN(B-A)
```

ABS

Calcula el valor absoluto de un número.

Formato: ABS (<número>)

Ejemplos:
```
100 MODU=ABS(D)
120 PRINT ABS(Z)
```

EL AZAR

En muchos juegos y en ciertos programas de simulación, el "azar" tiene una gran importancia. En el juego del parchís es fundamental el uso de un dado. En una simulación ha de contemplarse la variación aleatoria de algún factor (temperatura, viento...). Esta componente aleatoria se puede simular perfectamente en el ordenador.

El problema consiste en obtener una secuencia de números que no sigan (aparentemente) una regla fija. A cada uno de dichos números se le puede asignar, posteriormente, la cara de una moneda o de un dado, etc. De esta forma será posible evaluar los resultados de las distintas tiradas.

Una posible forma de calcular una secuencia "pseudoaleatoria" puede ser la siguiente:

1. Se parte de un número del 1 al 10.
2. Se eleva al cuadrado.
3. Se toma, del cuadrado, el dígito de mayor peso.
4. Se vuelve al paso 2.

Aplicando este método se obtendrán números de una cifra, sin una aparente relación entre sí. Véase, por ejemplo, la secuencia resultante a partir del número 9:

9 (9×9=81 − 8)
8 (8×8=64 − 6)
6 (6×6=36 − 3)
3 (3×3=9 − 9)
9 ...

La sencillez de este método trae consigo varios problemas. En el ejemplo se ve que a partir del cuarto número la secuencia se repite. El caso peor es cuando se parte del valor 1. Al elevarlo al cuadrado se obtiene de nuevo un 1 y, por lo tanto, la secuencia se repite con el mismo valor.

Existen métodos más complejos para el cálculo de secuencias "pseudoaleatorias". Muchos de ellos son fruto de la aplicación de teorías estadísticas y de probabilidad. Entre ellos se encuentra el método denominado "de las congruencias". Este se basa en el cálculo de números enteros de un número par de cifras (2n cifras).

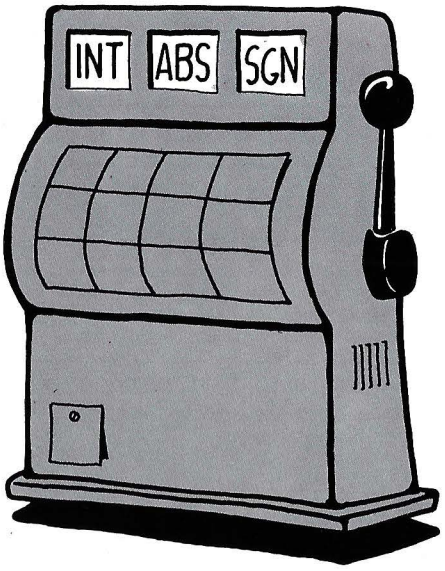
Partiendo de un número (que, además de cumplir esta condición, no ha de ser par ni terminado en 5), se calcula el siguiente multiplicándolo por una constante y truncando el resultado. La constante depende del número de cifras elegido. Así, para 2n cifras la constante valdrá 10↑n+3. Una vez truncado el resultado nos quedaremos con las 2n cifras menos significativas. El número resultante servirá de base para

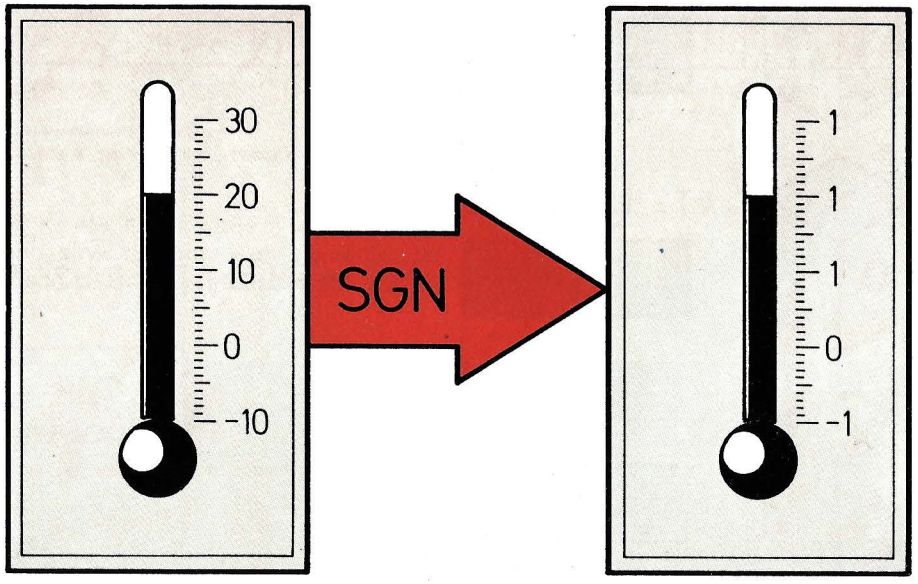
SGN

Obtiene 1 si el argumento es positivo, −1 si es negativo y 0 si es nulo.

Formato: *SGN(<número>)*

Ejemplos: 50 A=A+SGN(B)
70 PRINT SGN(N)





El dato entregado por la función SGN será 1 ó −1 según sea positivo o negativo el número que constituye su argumento.

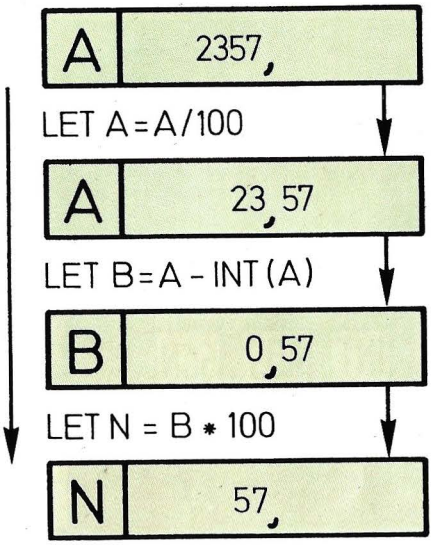


Ilustración del método empleado en el ejemplo del texto para obtener las cifras menos significativas de un número.

hallar el siguiente. Este método puede realizarse en BASIC de la forma que se expone a continuación.

```
10 REM AZAR
20 INPUT "NUMERO DE DIGITOS (PAR)";M
30 LET C=10 ↑ (M/2)+3
40 INPUT "NUMERO INICIAL";N
50 LET A=C*N
60 LET A=A/10 ↑ M
70 LET B=A-INT(A)
80 LET N=B*10 ↑ M
90 PRINT N
100 IF INKEY$="" THEN GOTO 100
110 GOTO 50
```

El número resultante de multiplicar C por N se trunca en las líneas 60 a 80. Para ello se hace correr la coma tantas posiciones a la izquierda como dígitos se desean truncar. Esto se consigue dividiendo por 10 ↑ M, siendo M el número especificado de dígitos. En la línea 70 se elimina la parte entera del resultado, con lo que únicamente quedan los M dígitos de menor peso. La línea 80 vuelve a colocar la coma en el lugar inicial, simplemente realizando la operación contraria a la efectuada en la línea 60.

Debido a la falta de precisión interna del ordenador, es posible que aparezcan números no enteros. Este problema puede solucionarse sustituyendo la línea 80 por esta otra:

```
80 LET N=INT(B*10 ↑ M)
```

La ejecución del referido programa, tomando 2 dígitos y como número inicial 17, conducirá, por ejemplo, a la siguiente pantalla:

```
RUN
NUMERO DE DIGITOS (PAR)?2
NUMERO INICIAL?17
21
72
35
55
14
82
...
```

Como se observa, la secuencia generada no sigue ningún orden lógico. Así pues, puede utilizarse para simular algún fenómeno aleatorio.

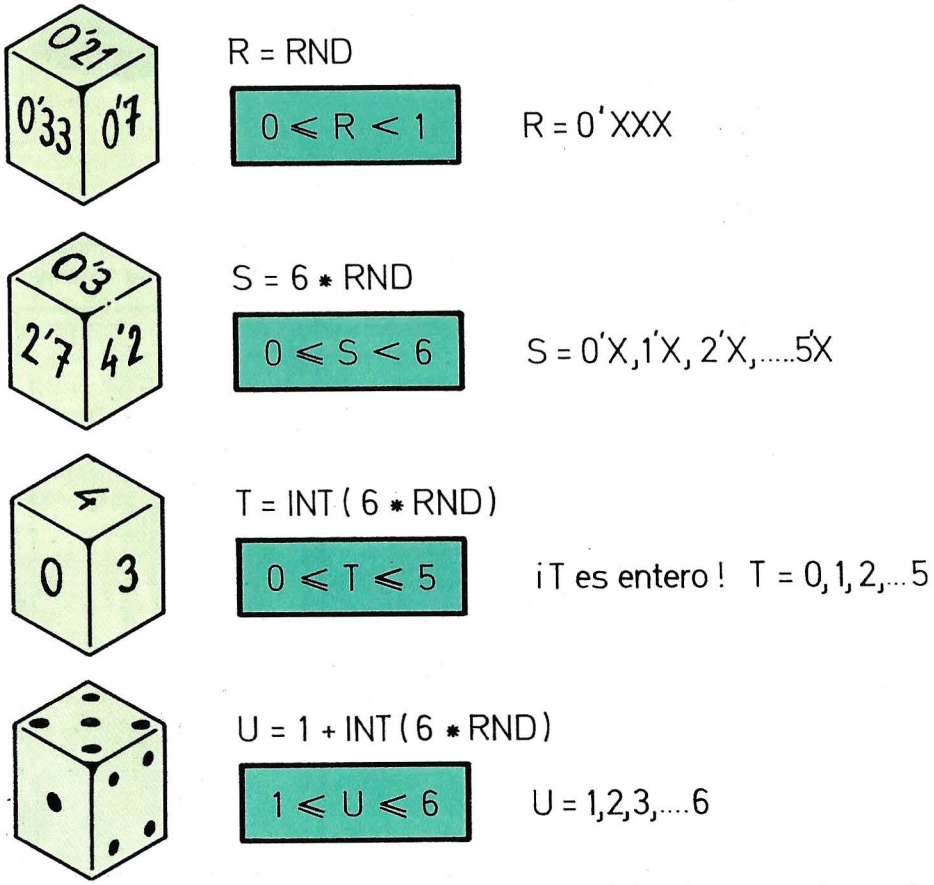
Cabe mencionar que el número de datos distintos va a depender del número de dígitos elegido.

La secuencia llega a un punto en que se vuelve cíclica: un conjunto de datos se repite indefinidamente. Si el ciclo de repetición incluye un conjunto grande de datos, dicha repetición apenas altera la aleatoria de los mismos.

En el ejemplo, con dos dígitos, tarde o temprano se llega a un grupo de unos 25 datos que se repiten cíclicamente. Aumentando el número de dígitos se incrementa también el ciclo de repetición.

En la amplia mayoría de los dialectos BASIC, existe una función adecuada para calcular números pseudoaleatorios. A esta función se accede por medio de la palabra clave RND (de RaNDom, aleatorio en inglés). Generalmente RND precisa de un dato de partida o "semilla" de la secuencia pseudoaleatoria.

En algunos aparatos, la función RND sólo calcula el siguiente número de la secuencia, partiendo de la semilla. En ellos la semilla debe especificarse entre paréntesis, a continuación de RND. De esta forma, siempre que se ejecute RND (<número>) y el <número> sea el mismo, se obtendrá idéntico resultado. Para encade-



Secuencia de operaciones a realizar para modificar el rango de los números generados a partir de la función RND, en orden a simular el lanzamiento de un dado.

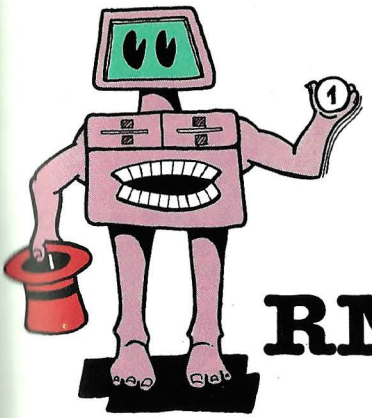
TABLA DE CONVERSION

ORDENADOR	RND	RANDOMIZE	INKEY$	GET	INT	ABS	SGN
	RND [(<c>)]	RANDOMIZE	INKEY$	GET <variable>	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
APPLE II (APPLESOFT)	RND (<c>)	RND(−<c>)	—	GET <variable>	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
APRICOT (M-BASIC)	RND [(<c>)]	RANDOMIZE	INKEY$	—	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
ATARI	RND (<c>)	—	—	—	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
CBM 64	RND (<c>)	RND (<c>)	—	GET <variable>	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
DRAGON	RND (<A>) (1)	—	INKEY$	—	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
EQUIPOS MSX	RND (<c>)	RND (−<c>)	INKEY$	—	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
HP-150	RND [(<c>)]	RANDOMIZE	INKEY$	—	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
IBM PC	RND [(<c>)]	RANDOMIZE	INKEY$	—	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
MPF	RND (<c>)	RND (−<c>)	—	GET <variable>	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
NCR DM-V (MS-BASIC)	RND [(<c>)]	RANDOMIZE	INKEY$	—	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
NEW BRAIN	RND	RANDOMIZE	—	—	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
ORIC	RND	RND (−<c>)	KEY$	GET <variable>	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
SHARP MZ-700 (MZ-BASIC)	RND (1)	RND (−<c>)	—	GET <variable>	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
SINCLAIR QL	RND [<A>[TO<B>]] (2)	RANDOMISE	INKEY$	—	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
SPECTRAVIDEO	RND (<c>)	RND (−<c>)	INKEY$	—	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
ZX-SPECTRUM	RND	RANDOMIZE	INKEY$	—	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)

<c>: Indica un número positivo cualquiera. No importa el valor de dicho número. <variable>: Nombre de variable de cadena alfanumérica. <núm.>: Constante, variable o expresión numérica.

(1) Calcula un número entero entre 0 y A. Si A vale 0 calcula un número entre 0 y 1.

(2) Genera un entero aleatorio entre A y B. Si sólo se especifica A, genera un entero entre 0 y A.



RND

nar los resultados y que el número calculado sirva de semilla al siguiente, se puede hacer algo semejante a lo que sigue:

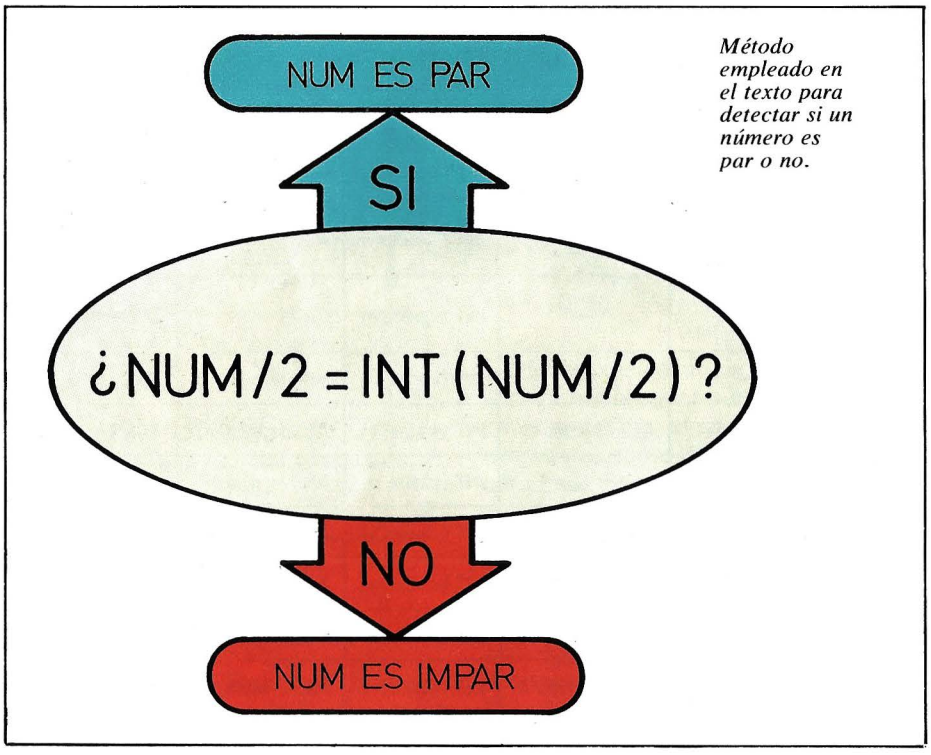
```
10 INPUT SEM
20 SEM=RND(SEM)
30 PRINT SEM
40 GOTO 20
```

RND

Genera un número aleatorio comprendido entre cero y uno.

Formato: *RND[(<número>)]*

Ejemplos: A=RND(1)
B=RND



Método empleado en el texto para detectar si un número es par o no.

Con ello se consigue una secuencia en la que cada nuevo número sirve de semilla al siguiente. En un programa complejo se utilizaría SEM como número aleatorio y se volvería a realizar el cálculo de forma análoga a la línea 20.
En ciertos dialectos BASIC la función RND no utiliza argumento, o carece de importancia el argumento que se le proporcione. En esos casos, dicha función se encarga por sí misma de guardar y reutilizar el último número generado. Para inicializar la semilla se dispone, entonces, de un comando adjunto: RANDOMIZE, en cuyo argumento se indica la semilla que ha de iniciar la secuencia.
Sea de una o de otra forma, la función RND dará, por lo general, un número decimal comprendido entre 0 y 1, pero que nunca llegará a alcanzar 1 (será un dato del tipo 0, XYZ).

UTILIZACION DEL AZAR

Haciendo uso de la función RND se puede simular cualquier fenómeno de tipo aleatorio; por ejemplo, el lanzamiento de un dado. Veamos cómo es posible construir un programa BASIC que calcule la puntuación asociada a la tirada de un dado.
El corazón del programa será, en efecto, la función RND. El único problema reside en que dicha función calcula un número comprendido entre 0 y 1. Así pues, lo primero que hay que hacer es convertir dicho número en un entero menor que siete y mayor que cero. El primer paso consiste en multiplicar el número aleatorio por 6.

De esta forma, los posibles datos estarán comprendidos entre 0 y 6; el número resultante podrá ser: cero con algo, uno con algo, ..., hasta cinco con algo. Está claro que lo que interesa es la parte entera de este resultado. El segundo paso será, pues, hallar la parte entera mediante la función INT.
Al aplicar INT se obtiene un número entero comprendido entre el cero y el cinco. Ahora sólo falta sumar una unidad para obtener un número entero entre 0+1=1 y 5+1=6. Este será el aspecto del programa BASIC.

```
10 REM DADO
20 INPUT "SEMILLA";S
30 RANDOMIZE S
40 PRINT 1+INT(6*RND)
50 GOTO 40
```

La línea 40 es la que realiza los cálculos explicados más arriba. En general, para obtener un número entero comprendido entre N (mínimo) y M (máximo), ambos inclusive, la fórmula a aplicar es la siguiente:

```
DATO=N+INT((M-N+1)*RND)
```

Si se tiene en cuenta que RND proporciona números "equiprobables" (esto es: en cualquier momento todos los números posibles tienen la misma probabilidad de aparecer), se puede construir un "lanzador de monedas" con la simple instrucción:

```
IF RND>0.5 THEN PRINT "CARA" ELSE PRINT "CRUZ"
```

Ello también se puede conseguir trasladando el rango de los resultados de RND, como en el caso del dado.
Otro método alternativo consiste en la puesta en práctica de la función SGN.

```
MONE=SGN(RND-0.5)
```

Con este último ejemplo se obtienen los valores 1 ó −1 con igual probabilidad. El contenido de MONE puede ser utilizado posteriormente para tomar una decisión aleatoria. En el siguiente ejemplo se consigue un aumento o disminución, de forma aleatoria del contenido de DATO.

```
DATO=DATO+SGN(RND-0.5)
```

En algunos intérpretes BASIC, la función RND da directamente un número entero. En tal caso, el rango de posibilidades se especifica en el argumento de RND.

RANDOMIZE

Introduce una "semilla" para generar una nueva secuencia aleatoria.

Formato: *RANDOMIZE[<semilla>]*

Ejemplos: 100 RANDOMIZE
50 RANDOMIZE 7

Pascal (1)

Un lenguaje de alto nivel estructurado y modular

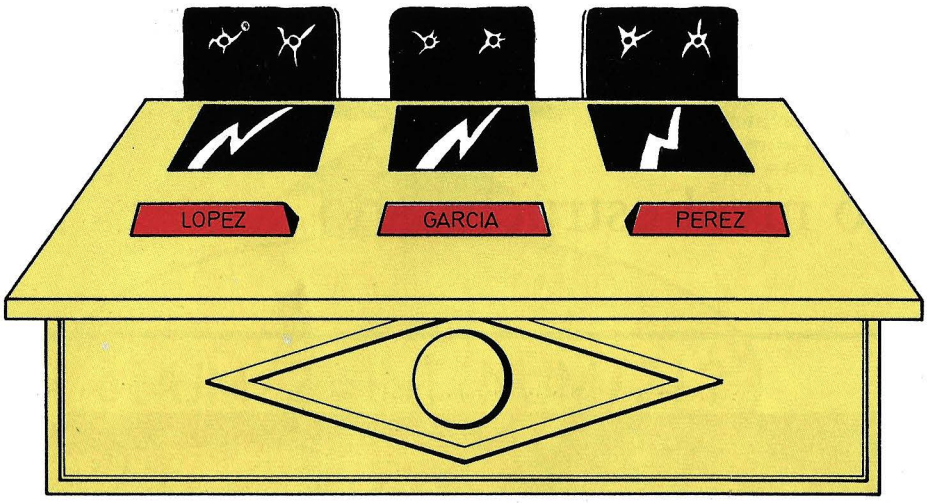
El PASCAL es un lenguaje de programación de alto nivel, modular y estructurado. Ha de utilizarse siguiendo una serie de reglas muy precisas, que se detallarán en este primero y en sucesivos capítulos.

Así como todos los lenguajes humanos incorporan un conjunto de reglas gramaticales, también existen reglas a las que hay que ajustarse para poder dialogar con la máquina en PASCAL.

Para que un programa sea fácilmente comprensible y susceptible de ser modificado con posterioridad, debe ser claro, inteligible y corto. Estos requisitos son fáciles de conseguir en un programa escrito en PASCAL, debido a las propias características de este lenguaje. En PASCAL, los programas se escriben de forma secuencial y ordenada. El empleo de bifurcaciones incondicionales (instituciones que rompen la secuencia de ejecución del programa) es desaconsejable, ya que dificultan el análisis y seguimiento del mismo. Al evitar su presencia —lo cual es siempre posible—, se logrará una ejecución ordenada de las instrucciones. Por lo demás, un programa en PASCAL se puede dividir en pequeños trozos o módulos que otorgan una gran flexibilidad al programa en orden a futuras modificaciones.

El PASCAL es un lenguaje informático de alto nivel, modular y estructurado.

Un programa en lenguaje PASCAL consta de tres bloques principales que aparecen dentro del mismo en un riguroso orden.



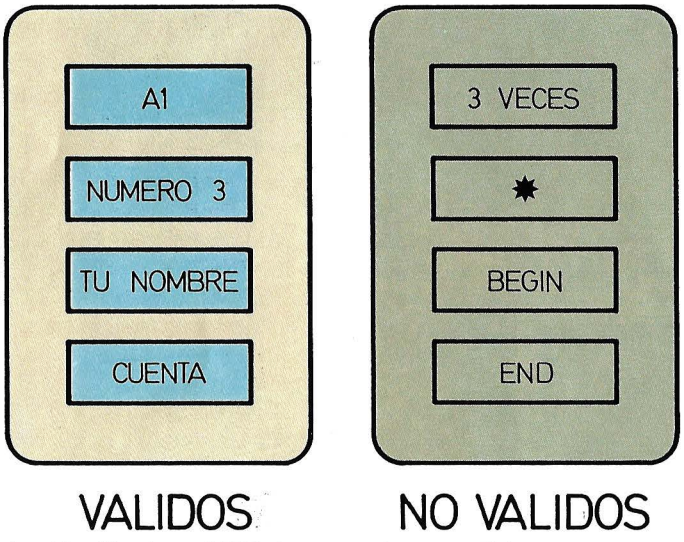
Los identificadores sirven para asignar nombres a las variables y, de esta forma, permitir su uso posterior dentro del programa.

una sola vez a lenguaje máquina y no instrucción a instrucción, a medida que las va procesando.

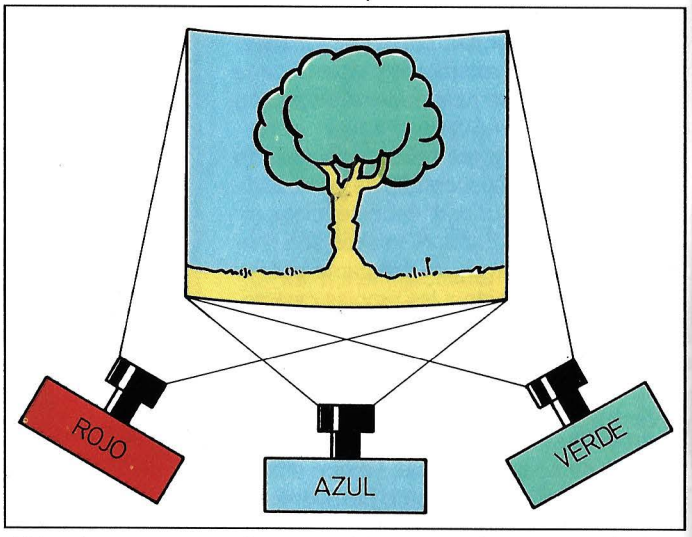
El formato de escritura de un programa es libre, de tal forma que las sentencias pueden colocarse en cualquier parte del listado. La única restricción a esta libertad es que cada instrucción debe terminar con un punto y coma para diferenciarla de la siguiente.

Los espacios en blanco se pueden insertar libremente, con lo que el listado del programa gana en claridad y legibilidad, al poder utilizar márgenes variables para cada módulo o porción del programa. En terminología informática, este método se llama *indentación* o profundidad lógica.

Las instrucciones se escriben en una notación similar al inglés restringido y al álgebra. Para ello se dispone de los siguientes

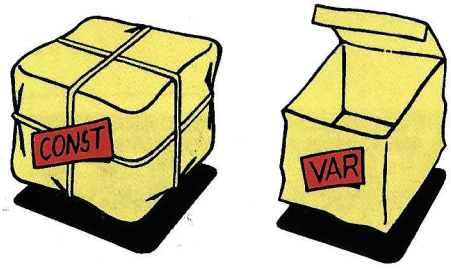


Los identificadores deben ajustarse a ciertas condiciones; por ejemplo, no son utilizables como identificadores las palabras reservadas del lenguaje PASCAL y las palabras que empiecen por una cifra numérica.



Al igual que ocurre con las composiciones cromáticas, partiendo de tres únicas estructuras (secuencial, de bifurcación e iterativa), pueden construirse programas PASCAL adecuados para resolver las más diversas aplicaciones.

Otra característica básica del PASCAL es la de ser un lenguaje estructurado. Del mismo modo que a partir de los tres colores primarios —rojo, verde y azul—, mezclándolos adecuadamente, es posible conseguir toda la gama del espectro; así también, partiendo de tres simples estructuras, a saber: secuencial, de bifurcación y repetitiva, se puede escribir cualquier programa.



En la zona de declaraciones y definiciones de un programa PASCAL, hay que diferenciar claramente los datos constantes y los datos variables.

El concepto de estructuración se refiere tanto al programa en sí, como a los datos del mismo. Estos últimos pueden estructurarse para reducir la complejidad de los algoritmos de trabajo, aumentar la claridad del programa y, en definitiva, conseguir un mayor rendimiento.

El programa en PASCAL siempre debe ser compilado antes de proceder a su ejecución. Ello significa que se traduce de

caracteres reconocibles por el ordenador (letras en mayúsculas o minúsculas):

a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 + − * / = : . , ( ) ' < > ; [ ]

Hay que destacar la distinción clara y precisa que existe entre, por ejemplo, los paréntesis y los corchetes; cada carácter tiene sus aplicaciones concretas y en ellas no puede ser sustituido por otro carácter similar.

Para incluir en el listado del programa los comentarios pertinentes, es preciso escribir el texto en cuestión encerrado entre paréntesis y asteriscos: "(*" y "*)". (*A que así es más fácil*). Desde luego, los

comentarios pueden emplazarse en cualquier parte del programa

IDENTIFICADORES

Cuando nacemos, nuestros padres nos conceden el privilegio de otorgarnos un nombre, por el que se nos puede identificar el resto de nuestra vida. De forma similar, los datos y elementos de un programa en PASCAL pueden recibir un nombre o *identificador,* por el que serán reconocidos en lo sucesivo dentro del programa.
El identificador proporciona, además, una idea del tipo de dato de que se trata y de su aplicación. Por ejemplo, si el dato coincide con la cantidad de vino producida en una determinada cosecha, puede otorgarse a este dato el nombre: VINO. Estos identificadores deben cumplir ciertas reglas, por ejemplo, deben empezar por una letra, aunque sus restantes caracteres pueden ser letras o cifras decimales. Su longitud no está limitada, si bien, el compilador sólo reconocerá los ochos primeros caracteres. Por ejemplo el identificador HIPOPOTAMOS, bajo la perspectiva del compilador, coincide con HIPOPOTAMAS.
Por otro lado, existen palabras con significado propio para el compilador que no se puede utilizar como identificadores. Estas palabras reservadas reciben el nombre de *delimitadores* y tienen reservada una misión específica; por ejemplo, PROGRAM,BEGIN,IF,END...

ASPECTO DE UN PROGRAMA EN PASCAL

Un algoritmo (método de resolver un problema determinado) escrito en PASCAL, consta de dos zonas fundamentales:
- La descripción de las acciones a realizar, coincidente con las sentencias del programa.
- La descripción de los datos a utilizar que son las declaraciones y las definiciones de parámetros.

Breve reseña histórica

Resulta fácil deducir que el nombre de PASCAL, otorgado a este lenguaje, procede del insigne matemático, físico, filósofo y escritor francés Blas Pascal; quien ya en 1641, a sus escasos 18 años inventó una máquina de calcular.
Los orígenes del PASCAL como lenguaje de programación se deben a un grupo de trabajo dirigido por Nicklaus Wirth.
Partiendo de las versiones 68 y W del ALGOL, trataron de obtener un lenguaje de alto nivel, muy sistematizado y pedagógico, adscrito a las tendencias de la programación estructurada.
En 1970 se presentó el primer diseño de compilador. Posteriormente el PASCAL se fue afianzando hasta llegar a estandarizarse en 1978. Actualmente, forma parte, por derecho propio, de los lenguajes habituales en el mundo del software.

Aspecto general de un programa

```
PROGRAM RAICES2 (INPUT,OUTPUT); (*CABECERA*)

CONST DOS=2; (*DECLARACIONES Y DEFINICIONES*)
VAR A,B,C.RAIZ1,RAIZ2,IMAG,PREAL,DISCRI:REAL;

BEGIN (*CUERPO PRINCIPAL*)
      READ (A,B,C)
      IF A=0 THEN WRITE ('ERROR')
      ELSE
      BEGIN
            DISCRI:=SQR(B)-4*A*C;
            IF DISCRI<0 THE
                BEGIN
                      DISCRI:=-DISCRI;
                      IMAG:=SQRT(DISCRI)/(DOS*A);
                      PREAL:=-B/(DOS*A);
                      WRITELN('RAIZ1=', PREAL,'+J', IMAG);
                      WRITELN('RAIZ2=',PREAL,'-J',IMAG)
                END
            ELSE
                BEGIN
                      RAIZ1:=(-B+SQRT(DISCRI))/(DOS*A);
                      RAIZ2:=(-B-SQRT(DISCRI))/(DOS*A);
                      WRITELN('RAIZ1=',RAIZ1);
                      WRITELN('RAIZ2=',RAIZ2)
                END
      END
END.
```

Un programa escrito en lenguaje PASCAL se puede dividir en tres zonas fundamentales:

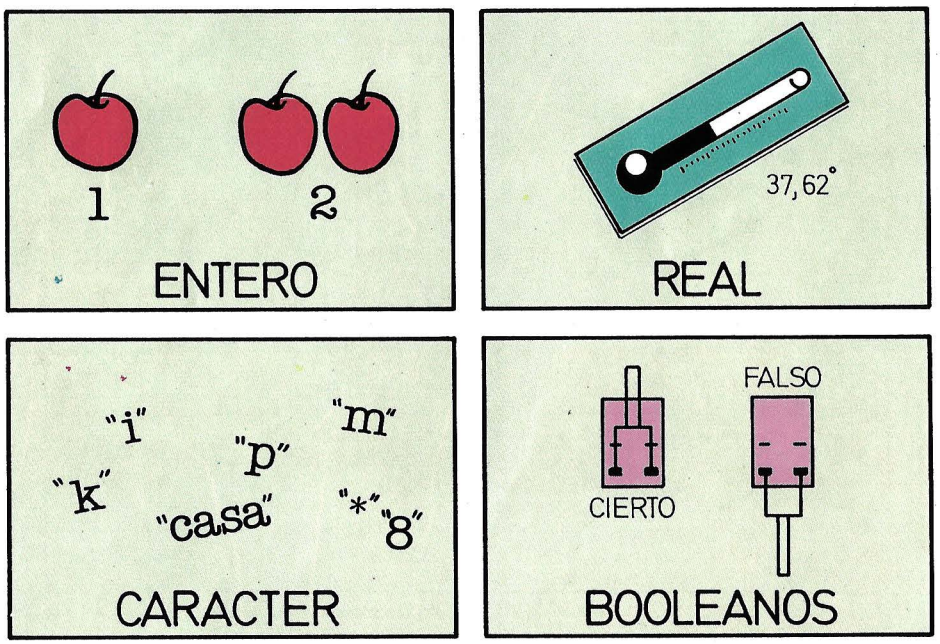
- *La cabecera* que da nombre al programa y le asigna una lista de parámetros. Estos coinciden con los nombres genéricos de los ficheros que se utilizan en el programa para el intercambio de información con el usuario (entrada y salida de datos).
- *Declaraciones y definiciones.* Zona en la que se reseñarán todas las etiquetas, constantes, tipos, variables, procedimientos y funciones a utilizar en el resto del programa.
- *El cuerpo del programa* que, como su nombre indica, constituye el conjunto de instrucciones ejecutables que realizarán la función asignada al programa.

En el ejemplo se observa la puesta en práctica del referido *formato libre;* en efecto, la indentación contribuye a mejorar la presentación del programa.
El formato libre implica además la presencia de un separador de instrucciones, coincidente con el signo punto y coma (;). Puede conducir a confusión el hecho de que el ejemplo incluya sentencias que no acaban en punto y coma.
Ello es correcto y se debe a que alguna de ellas conforman una única instrucción, como se verá más adelante; en consecuencia, sería un error fraccionarlas con algún punto y coma.

TABLA DE COMANDOS-PASCAL

Instrucción	Cometido	Observaciones
PROGRAM <id.>(<par.>);	Asignación de parámetros de acceso a un programa	Cabecera del programa
BEGIN <ins.>[,<ins.>,...]END;	Delimita bloques de instrucciones ejecutables	Cuerpo del programa
CONST <id.>=<valor>; [<id.>=<valor>;] [...]	Definición de constantes	Parte de las declaraciones y definiciones
VAR <id.>[,<id.>,...]: <tipo>; [...]	Declaración de variables y su tipo	Parte de las declaraciones y definciones

<id.>: identificador válido PASCAL. <par.>: lista de parámetros de los ficheros que utilizará el programa. <ins.>: instrucción ejecutable. <valor>: valor numérico o literal constante. <tipo>: uno de los tipos de datos normalizados o previamente definidos en PASCAL.



Las constantes y las variables PASCAL pueden encuadrarse en tres categorías o "tipos de datos": enteros, reales, carácter y booleanos.

A su vez, un programa en PASCAL está dividido en tres bloques independientes:

— Cabecera.
— Declaraciones y definiciones.
— El cuerpo de sentencias.

La *cabecera* sirve pera identificar al programa. Consta de la palabra reservada PROGRAM, seguida por un identificador válido PASCAL. Este último coincidirá con el nombre genérico que se le adjudique al programa.
A continuación, y encerrados entre paréntesis, figurarán los nombres de los ficheros o dispositivos lógicos a través de los que el programa se comunicará con el exterior. Estos ficheros son propios de cada sistema. Generalmente se llama INPUT al fichero de entrada (teclado) y OUTPUT al fichero de salida (impresora o pantalla).

La zona de *declaraciones y definiciones* es opcional y comprende las siguientes partes que, en el caso de que existan todas ellas, deben seguir el orden riguroso en el que se relacionan a continuación:
1. Declaraciones de etiquetas.
2. Definición de constantes.
3. Definición de tipos.
4. Declaración de variables.
5. Declaración de procedimientos y funciones.

Cada una de estas cinco partes se definirá con detalle en un próximo capítulo.
El cuerpo de sentencias incluye a las sentencias ejecutables del programa; sentencias que quedarán encerradas por las palabras reservadas BEGIN y END. Tras el END aparecerá un punto final ineludible.

LOS DATOS DEL PASCAL

Los datos son la materia prima con la que el ordenador va a elaborar su producto que no es otro que el resultado de la ejecución del programa. Los datos pueden aparecer en forma de valores *constantes* o representados por medio de *variables,* estos últimos pueden ver alterado su valor durante la ejecución del programa. Una semana tiene siete días, éste es un dato constante; mientras que el dinero de nuestro bolsillo suele ser un dato "bastante" variable.
Los datos se tienen que declarar en la zona de declaraciones y definiciones del programa, para que sea factible su uso posterior. Para declarar una constante se utiliza la siguiente notación:

CONST <identificador> = <constante>;

Análogamente, las variables se declaran con la notación que sigue:

VAR <identificador1> [,<identificador2>,...]: <tipo>;

El argumento *tipo* se refiere a la clase de variable de que se trata y sirve para indicar el conjunto de valores que la variable puede asumir. Por lo tanto, la declaración de una variable define inmediatamente su tipo, con la precisión a señalar de que cada variable PASCAL sólo puede ser de un único tipo.

MS/DOS (3)

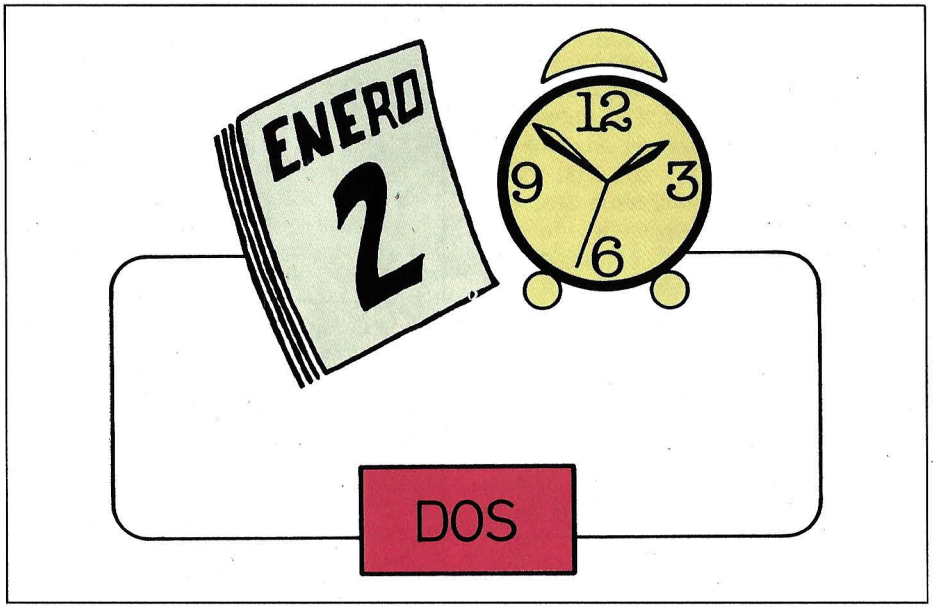
Comandos básicos del DOS 1.0

Una vez introducida la filosofía general del MS/DOS, y detallados los aspectos básicos de la versión DOS 1.0, llega el momento de presentar los comandos que brinda este sistema operativo.

Los comandos que se describen en el presente capítulo son los más comunes y de uso más frecuente. Posteriormente, se acometerá el estudio de los restantes comandos, adecuados para ejecutar tareas más complejas.

Los comandos que van a ser objeto de análisis en los próximos párrafos, son comunes a las versiones 1.0 y 2.0 del sistema operativo MS/DOS. A pesar de su analogía, cabe señalar que en la versión más avanzada (DOS 2.0), éstos tienen mayor potencia en algunos casos. Tal circunstancia quedará reflejada en el texto.

Un detalle a precisar de antemano es que cada fabricante, aun adoptando la pauta general, introduce pequeñas diferencias en los nombres y, en ocasiones, en la sintaxis de ciertos comandos. Ello exige un repaso del manual de usuario específico del ordenador que vaya a utilizarse.

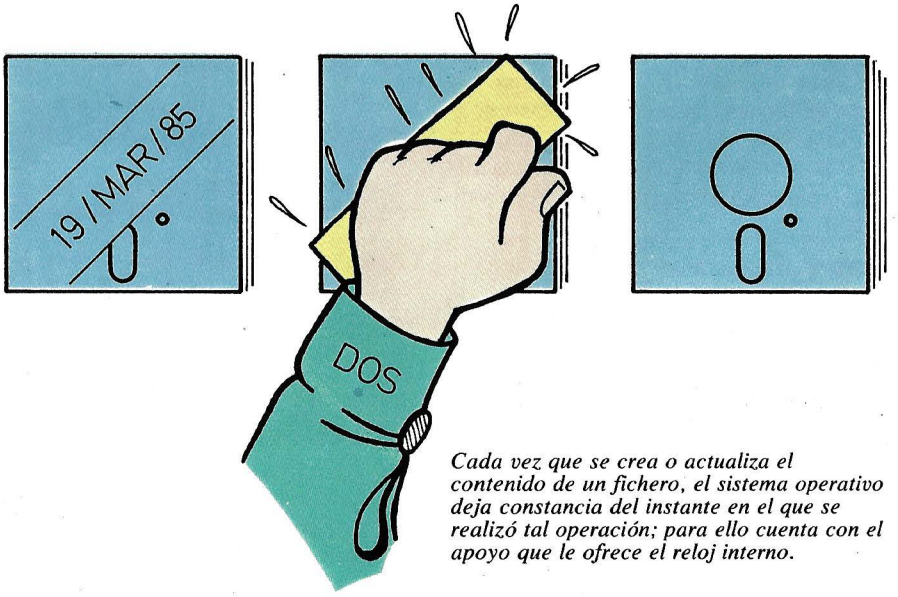


El MS/DOS cuenta con un reloj interno cuya función básica es suministrar información al sistema. Su manipulación se realiza por medio de los comandos DATE y TIME.

COMANDOS DE CONTROL DE TIEMPO

El sistema operativo MS/DOS ofrece al usuario la posibilidad de operar con un reloj interno. La función básica de dicho reloj es suministrar información al sistema operativo, de tal forma que cuando se crea o actualiza algún tipo de información sobre un disco, queda constancia del instante en el que se realizó esta operación.

La referida indicación es sumamente importante, toda vez que suministra al usuario una completa información acerca de



Cada vez que se crea o actualiza el contenido de un fichero, el sistema operativo deja constancia del instante en el que se realizó tal operación; para ello cuenta con el apoyo que le ofrece el reloj interno.

cuándo fue la última vez que se actualizaron ciertos datos. Ello le permitirá conocer sobre qué información se trabajó y en qué período de tiempo exactamente. Sin lugar a dudas, dicha posibilidad facilita el control sobre el uso de los datos. Los comandos que permiten acceder al reloj interno son DATE y TIME, cuyas respectivas funciones son:

DATE: Presentar y actualizar la fecha

TIME: Presentar y actualizar la hora

La presentación de la fecha y la hora parece una tarea bastante simple, y de hecho es así. La posibilidad de actualización deriva de que el reloj interno es volátil y, en consecuencia, ha de ser actualizado cada vez que se conecta el ordenador. Tras finalizar el proceso de arranque en frío y antes de iniciar el proceso de operación normal, el propio sistema operativo interroga al usuario respecto a la fecha y hora actuales. Mientras el ordenador no sea desconectado, el reloj seguirá funcionando y será capaz de pasar de un día a otro, así como de detectar si el año en curso es bisiesto con el fin de introducir las oportunas correcciones.
Para operar con ambos comandos, caben dos modalidades:
— Tecleando el nombre del comando y pulsando la tecla de retroceso de carro (RETURN, ENTER o <CR>).
En este caso, el ordenador presenta en pantalla la fecha y hora actuales, y aguarda a su posible modificación. Si se pulsa <CR> directamente, el dato no sufrirá alteración alguna. Por el contrario, de introducir una nueva fecha u hora, ésta será almacenada en memoria al accionar <CR>, y el ordenador empezará a contar el tiempo desde ese instante.
— Tecleando el nombre del comando y el dato de fecha u hora correspondiente.
En esta modalidad se produce la actualización de forma indirecta, sin realizar la máquina ningún tipo de consulta. Por ejemplo:

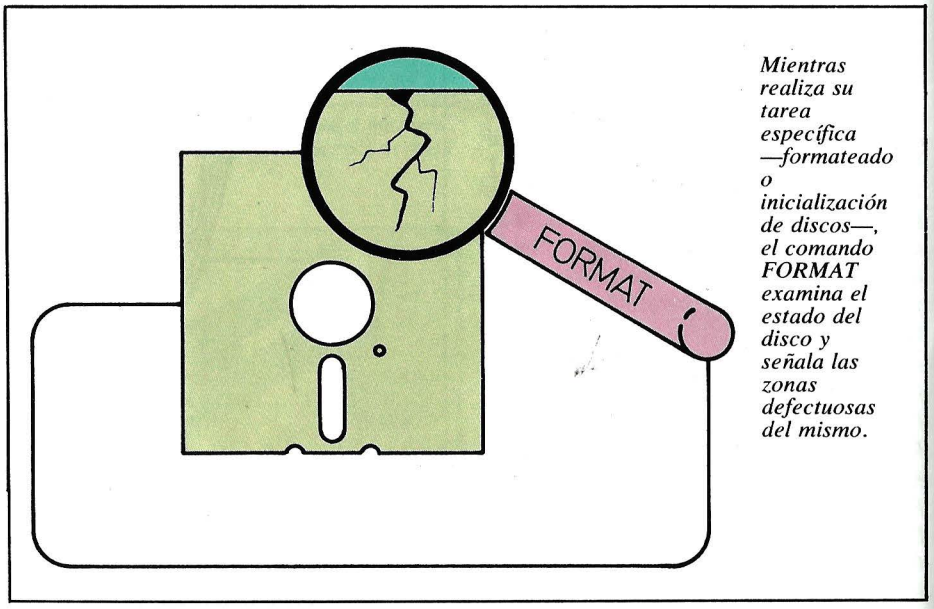
DATE 23-01-84 <CR>

COMANDOS DE PREPARACION DE DISCO

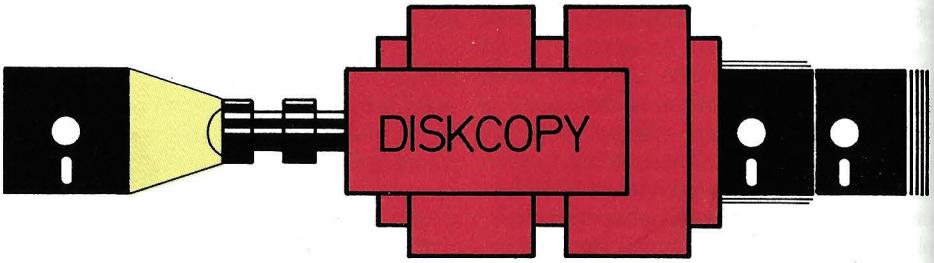
Dada su orientación a la actividad de gestionar archivos en disco, el MS/DOS pone a disposición del usuario una serie de herramientas destinadas a la preparación de los disquetes. Estas adoptan la forma de los comandos externos FORMAT, SYS y DISK-COPY.

FORMAT

La misión del comando FORMAT es la de inicializar (formatear) un disquete virgen. Sobre éste crea una virtual "guía electrónica" que permitirá al sistema operativo trabajar posteriormente con el mismo. Durante esta operación, el propio comando examina el disquete, indicando los posibles defectos que este pudiera tener. En el caso de existir algún defecto, FORMAT lleva a cabo dos funciones.
— Advertir al usuario de la presencia de los defectos, indicando el tamaño en bytes de las zonas defectuosas, así como la cantidad total de memoria utilizable en el disco.
— Marcar el área defectuosa por medio de unos indicadores. Esta no será utilizable, aunque sí será posible utilizar el resto del disco para almacenar datos.
Lo normal es que el disco se formatee sin mayores problemas. De no ser así, y si el número de defectos fuera apreciable, será preciso rechazar el disco en cuestión. Posiblemente el disquete sea de fabricación defectuosa, o bien puede existir algún problema con la cabeza grabadora de la unidad de dico.
Una posibilidad que permite el comando FORMAT es la de incluir los ficheros que contienen en sistema operativo en el disquete que se está formateando. Con ello, el nuevo disco podrá utilizarse para el arranque en frío del ordenador sin necesidad de recurrir a un disquete especial. Tal posibilidad la abren los interruptores "/S" o "−S", dependiendo del modelo de ordenador utilizado.
Otros interruptores permiten tener en cuenta el tipo de disco en uso (si es de simple o doble densidad) en orden a realizar un formateado correcto.
FORMAT es un comando cuyo empleo



*Mientras realiza su tarea específica —formateado o inicialización de discos—, el comando **FORMAT** examina el estado del disco y señala las zonas defectuosas del mismo.*



*El comando **DISKCOPY** permite obtener copias exactas de la información almacenada en un disquete.*

puede resultar peligroso, toda vez que puede actuar sin problema alguno sobre un disco útil, repleto de datos. El resultado será la pérdida de todos ellos. En consecuencia, su puesta en práctica exige la adopción de las debidas precauciones.

SYS

El comando SYS se encarga de transferir el sistema operativo DOS a un disquete ya formateado. La necesidad de ejecutarlo puede responder a uno de los siguientes motivos:

— El uso de un programa que exige la presencia de una copia del sistema operativo en el disco que almacena el programa.

— La aparición de una nueva versión del DOS. Este comando permitirá introducirla en aquellos disquetes que contengan la versión antigua, sin dañar los ficheros de datos que pudieran estar almacenados.

Hay que precisar que este comando no realiza la copia de un fichero del sistema operativo denominado COMMAND.COM. Este debe copiarse de forma independiente, por medio de la orden COPY.

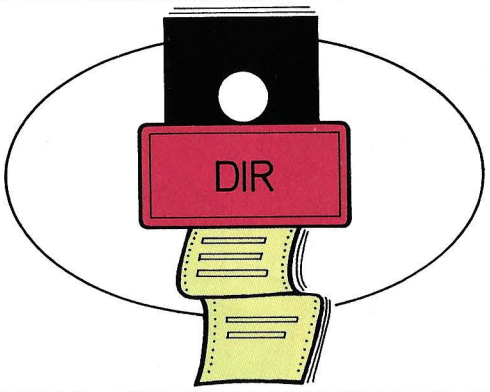
DISKCOPY

Su cometido es copiar el contenido de un disco íntegramente en otro, sin ningún tipo de operación intermedia. El resultado es una reproducción exacta del disquete original. Hay que hacer notar, sin embargo, que en algunas versiones del sistema operativo este comando no reproduce el formateado; de ahí que sea necesario inicializar previamente el disco que va a recibir la copia.

Cuando se desea transferir íntegramente los datos de un disco a otro y el disquete de origen se encuentra totalmente lleno, este comando se revela como sumamente eficaz. A pesar de su gran efectividad, hay algunas precisiones que conviene señalar:

— El comando DISKCOPY no admite en ninguno de los dos discos implicados en la operación de copia la existencia de algún defecto de los que el comando FORMAT detecta y señala.

— No optimiza los resultados del disco. En consecuencia, si en el disquete original hay pocos datos y están distribuidos esporádicamente en su superficie, esta misma distribución será la que aparezca en la copia.



Dentro del grupo de comandos adecuados para la "administración del disco" se encuentra DIR; su cometido es proporcionar una lista de los ficheros residentes en el disco que se especifique.

La comunicación ordenador-usuario

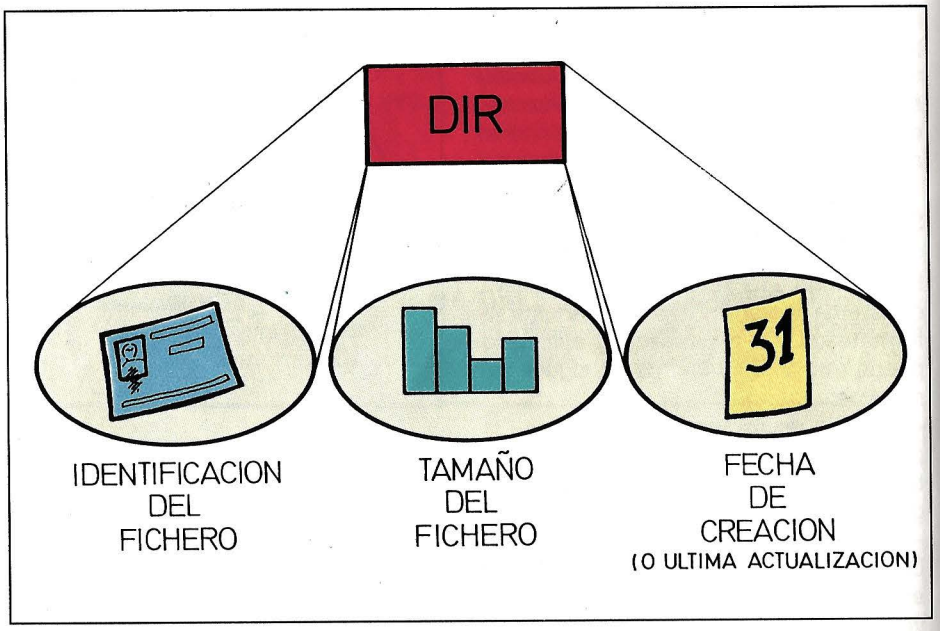
Cuando el usuario se sienta ante el terminal de un ordenador e inicia una tarea, da comienzo un proceso de intercambio de información con la máquina. El usuario introduce unos datos que el ordenador analiza en función de los dictados del programa. Como colofón, la máquina devuelve toda una serie de mensajes; información que el usuario considerará en orden a modificar los datos inicialmente introducidos para repetir de nuevo el proceso. Este diálogo proseguirá hasta que concluya la sesión de trabajo. Durante el período de actividad, el usuario necesita que la información procedente del ordenador le sea presentada con celeridad; independientemente del tiempo empleado por la CPU para el cálculo. Este hecho tiene un fuerte impacto en el diseño del ordenador, ya que determina la propia capacidad de los canales a través de los que dialogan el ordenador y el usuario. En efecto, si pensamos en un sistema de diseño gráfico, el usuario necesita que los diseños se visualicen inmediatamente en la pantalla. Como quiera que los trazados gráficos exigen un gran volumen de información y memoria, es preciso que los canales de comunicación sean de gran capacidad. Sin embargo, en tareas de programación, durante las que el ordenador solicita esporádicamente información del usuario, la capacidad exigida a los canales es notablemente inferior.

Una forma de medir estas necesidades es recurriendo al cociente entre la máxima información transmitida por unidad de tiempo, y el volumen medio de información transmitida en el transcurso de una sesión de trabajo. Una tarea de programación tendrá un cociente bajo, mientras que un proceso gráfico lo tendrá muy alto.

Tipo de aplicación	Velocidad media de transmisión (bits/sg)	Velocidad punta de transmisión (bits/sg)
Cálculo corriente, empleando el lenguaje BASIC	8	100
Control de almacenes utilizando una base de datos	10	150
Entrada de datos a través de formatos de pantalla	15	1 500
Reserva de billetes por ordenador	10	2 000
Sistema gráfico en el que se utiliza un lápiz óptico para crear los diseños	200	200 000

Existen grandes diferencias entre la cantidad de información transmitida, por término medio, en el transcurso de una sesión de trabajo, y la información a transmitir en un breve período de tiempo para prevenir retrasos inaceptables por el usuario.

COMANDOS PARA LA ADMINISTRACION DEL DISCO



La información entregada por la máquina en respuesta a la orden DIR, incluye datos relativos a la identificación de los ficheros que residen en el disco afectado, a su tamaño y a la fecha y hora de su creación o última modificación.

En esta categoría se integran dos comandos básicos: DIR y CHKDSK. Su función es proporcionar al usuario una idea de cuál es el contenido real del disquete sobre el que opera. En ciertas versiones del MS/DOS aparece un tercer comando de este tipo: CAT, con funciones muy similares a DIR.

DIR

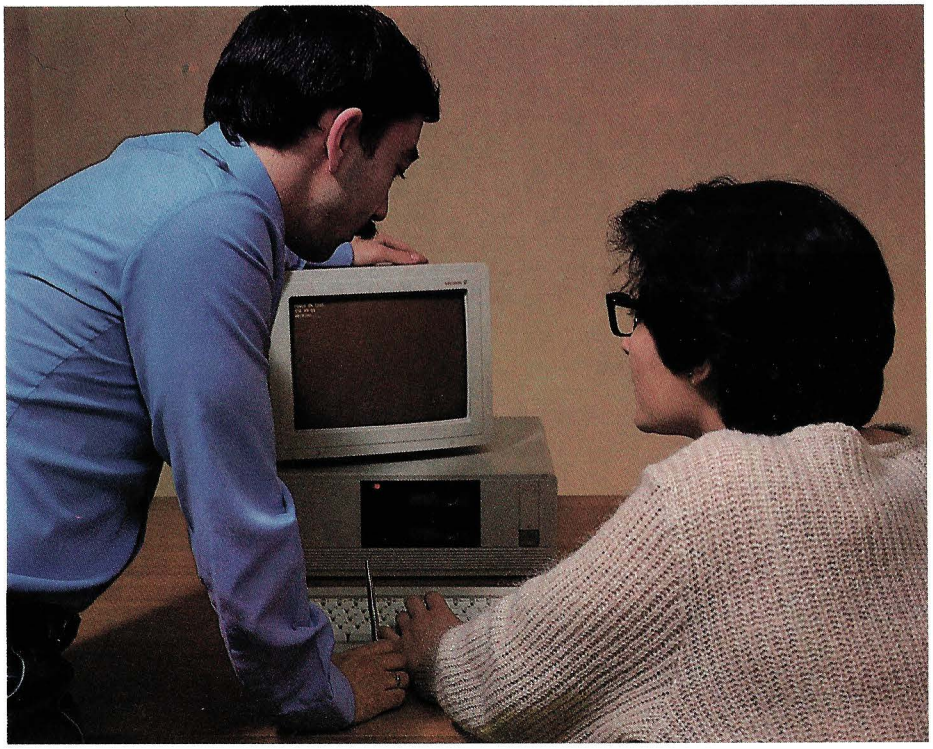
Suministra una idea completa de los ficheros contenidos en el disco. Sin embargo, no da ningún tipo de indicación acerca de la existencia de lo que se denominan "ficheros ocultos". Estos últimos contienen determinadas zonas del sistema operativo. El comando DIR puede adoptar distintos formatos:

DIR A:

Confecciona una lista con todos los ficheros residentes en la unidad de disco A, indicando, en cada caso, el nombre, extensión del nombre, longitud del fichero en bytes, así como la fecha en la que el fichero fue creado o actualizado por última vez.

DIR Nombre del fichero:

Suministra la misma información que en el caso anterior, sólo que restringida a un único fichero.
Normalmente, suelen asociarse al comando DIR algunos interruptores que permiten efectuar una pausa cada vez que se llena la pantalla ("/P" o "−P"), o eliminar la información acerca del tamaño y la fecha de creación del fichero ("/W" o "−W").



Los comandos para la administración del disco (DIR y CHKDSK) permiten al usuario obtener información inmediata y actualizada del contenido de los discos con los que opera.

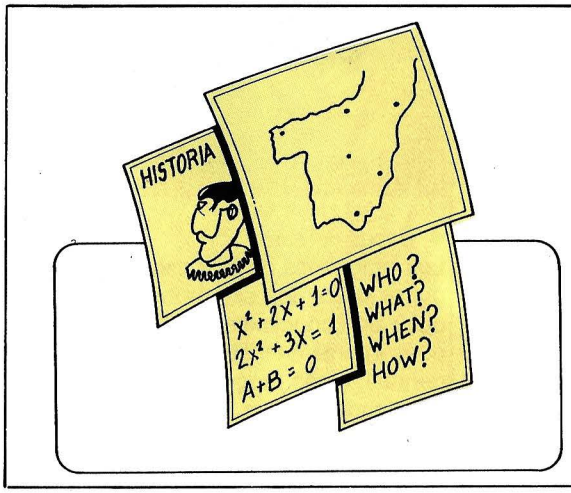
CHKDSK

Las funciones de este comando son similares a las del comando DIR, sólo que más condensadas. Al invocarlo, revela los siguientes datos:
Espacio libre en el disquete.
Espacio ocupado por ficheros.
Discrepancias en el empleo del espacio de almacenamiento.
Número de ficheros en el disquete.
Número de ficheros ocultos (si los hay).
Una de las funciones de CHKDSK es la de reparar cualquier defecto lógico en el uso del espacio de almacenamiento. En el caso del MS/DOS versión 1.0, tal operación se realiza de forma automática.

Aplicaciones lúdicas

Software de juegos y entretenimiento

La popularización de los ordenadores personales y la gran afición a los videojuegos han contribuido al nacimiento de miles de programas de aplicación dedicados a entretener al usuario. Al principio sólo se perseguía simular mediante un programa el funcionamiento de los videojuegos presentes en salones de recreo, cafeterías, bares, etc. Pronto, el desbordante ingenio de los programadores y la facilidad que supone programar en un lenguaje evolucionado como es el BASIC, permitió que el nivel de satisfacción e "inteligencia" de los juegos programados para microordenadores superara ampliamente al de los videojuegos tradicionales.



Uno de los apartados del software de juegos lo ocupan los programas de divertimento didáctico. Esta es una forma amena y entretenida de ofrecer al usuario la posibilidad de estudiar o repasar conocimientos relativos a cualquier disciplina.

CLASIFICACION DEL SOFTWARE DE JUEGOS

A pesar de que su carácter lúdico puede peyorizar a este tipo de programas, no cabe la menor duda que existen auténticas "obras de arte" dentro del software de juegos y entretenimiento.
De hecho, se puede comprobar estadísticamente que los ordenadores domésticos utilizados fuera del ámbito empresarial se dedican mayoritariamente a dos actividades: aprendizaje de programación y juegos; y no se puede despreciar algo que se impone a nivel popular. Con objeto de clasificar a los programas de juegos existentes, vamos a agruparlos en siete grandes grupos; más adelante, se describirán las características de cada uno de ellos.

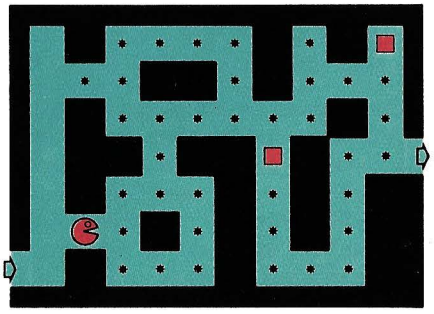
1. *Didácticos*
Una de las mejores formas de divulgar una disciplina es presentarla de forma amena y entretenida, con objeto de que el alumno se divierta a la vez que estudia. Existe una gran diversidad de programas que, aún bajo la estructura propia de un juego, obligan al usuario a dominar la materia tratada.

2. *Simuladores*
Es sorprendente cómo se pueden reproducir situaciones reales incluso en los más pequeños ordenadores. Los simuladores son útiles para realizar entrenamientos sin riesgos sobre un modelo extraído de la realidad, y por supuesto también para pasar ratos agradables.

3. *Laberintos*
Otro de los temas más explotados como juego programado son los laberintos. El objetivo del usuario consiste en encontrar un camino por el que escapar de un entorno complicado. En algunas variedades de laberintos, existe un problema adicional que consiste en eludir el ataque de férreos enemigos que impiden progresar con tranquilidad.

4. *Animación*
La principal atracción de los juegos de animación consiste, precisamente, en las evoluciones de los objetos en la pantalla. En la mayoría de los casos, el usuario controla los movimientos del dibujo animado mediante el teclado o por medio de algún mando especial (joysticks, paddles, ...).
Las técnicas de animación son también importantes en los juegos de simulación; no obstante, en estos últimos el hecho de tener que aferrarse a una situación real impide que las fantasías puedan ser tan abundantes como en los juegos de simple animación.

5. *Musicales*
Dentro de esta categoría cabe distinguir



En los juegos de tipo "laberinto", el usuario se ve obligado a encontrar la salida del mismo sorteando, en muchos casos, la dificultad adicional que supone la presencia de feroces atacantes.

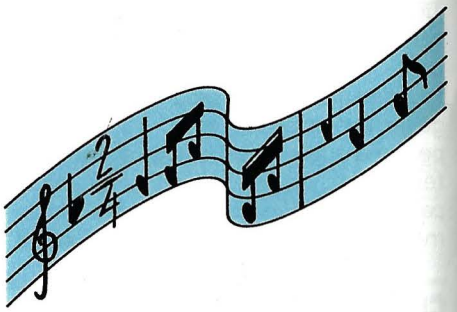
dos subgrupos. El primero agrupa a todos los juegos que, aun sin especializarse en el tema musical, lo utilizan como apoyo de su actividad básica. En el segundo subgrupo se incluyen los programas versados exclusivamente en el aspecto musical, permitiendo al usuario participar en la interpretación de una melodía o en la composición de la misma.

6. *Aventuras galácticas*
Desde luego, no hay que olvidar a los tradicionales programas de "marcianitos". Denominados popularmente así, en realidad se trata de juegos de invasión: unos seres extraños, generalmente de aspecto cambiante, descienden desde el extremo superior de la pantalla y amenazan con destrubir la nave controlada por el jugador. Este debe eludir su ataque e impedir su descenso utilizando el armamento a su destruir la nave controlada por el jugador. marcianitos se van haciendo cada vez más peligrosos hasta que, finalmente, consiguen destruir las armas del usuario.

7. *Varios*
En los seis grupos anteriores es posible encuadrar a la inmensa mayoría de los programas de juego existentes. No obstante, la inventiva humana es capaz de invalidar cualquier tipo de clasificación para estos programas. De ahí que sea necesario considerar un último grupo de programas variados en el que se incluirán todos los programas difíciles de clasificar según los criterios anteriores.

- *Bases de datos*

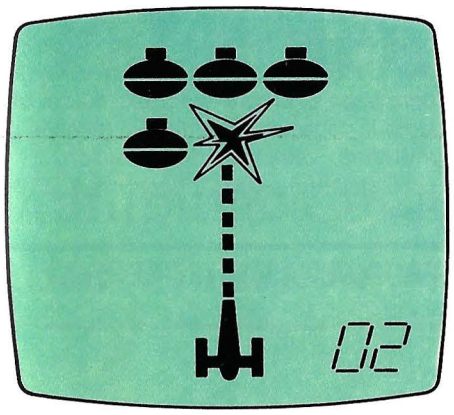
Los programas de tipo didáctico, y en algunos casos los simuladores, deben adquirir un conocimiento sobre el que después intentarán instruir a los usuarios. En general, estos programas deben incorporar algunas de las características de las bases de datos, con objeto de realizar búsquedas rápidas entre los registros que forman su "sabiduría".
Las materias tratadas dentro de los programas didácticos tienen, en algunos casos, un carácter marcadamente jerárquico. Por ejemplo, los programas dedicados a profundizar en la historia, suelen



Los programas musicales permiten convertir al ordenador en un verdadero instrumento musical, en muchos casos reflejando incluso la partitura en pantalla.

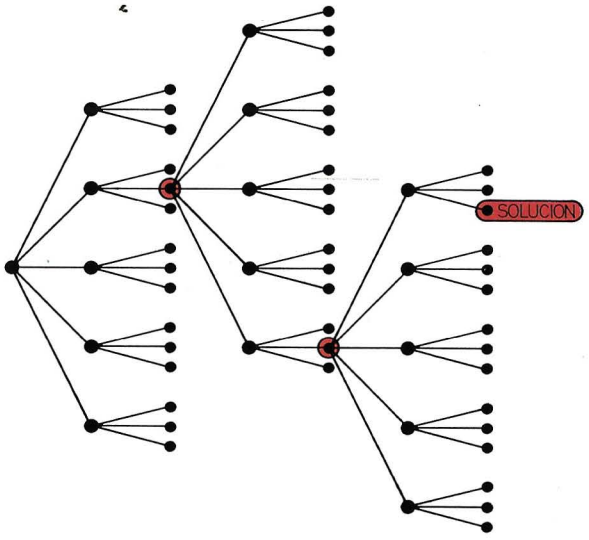
PRINCIPALES CARACTERISTICAS DE LOS PROGRAMAS PARA JUEGO

La complejidad de los programas de entretenimiento puede ser en algunos casos muy superior a la de los programas de aplicación convencionales. Con objeto de sobresalir dentro de la enorme oferta existente en el mercado, los programadores encargados de crear programas de juego de tipo convencional, tienen que aguzar el ingenio y utilizar sofisticadas técnicas de programación. A continuación se relacionan los principales puntos de apoyo para este tipo de programas.



Las aventuras galácticas constituyen, sin duda alguna, el grupo más popular de los programas de juego... ¿Quién no se ha lanzado alguna vez a la caza de escurridizos "marcianitos" pilotando una rápida nave galáctica?

disponer de información sobre los árboles genealógicos de las casas reales. Estos utilizan técnicas propias de las bases de datos jerárquicas. En cambio, existen otros programas didácticos con un marcado carácter relacional; normalmente, todos los programas del tipo pregunta-respuesta ponen en práctica las técnicas utilizadas en las bases de datos relacionales.



La figura representa el esquema selectivo de los programas inteligentes. Ante la imposibilidad de estudiar todas las opciones, el programa selecciona y trabaja exclusivamente con las más prometedoras.

● *Lenguaje en código máquina*

Algunos programas de entretenimiento, ya sea porque manejen mucha información o porque deban incluir efectos especiales, necesitan que el microprocesador ejecute sus instrucciones muy velozmente. Ante esta situación, no vale con utilizar como lenguaje de programación el BASIC. Normalmente, este lenguaje suele tener carácter "interpretable" en los ordenadores personales; esto es: antes de ejecutar cada una de las instrucciones incluidas en el programa, el microprocesador debe traducirlas a código máquina. En consecuencia, existen dos únicas soluciones para lograr que la ejecución sea más rápida.

— Conseguir un compilador que realice la traducción del programa una única vez, y a partir de ese momento ejecutar el programa ya traducido a código máquina.

— Utilizar directamente un lenguaje de bajo nivel, adecuado al código máquina del microprocesador utilizado en cada equipo.

● *Modelos matemáticos*

En muchos juegos, sobre todo en los de simulación, se debe utilizar un modelo matemático como sustento del "universo" que representan.

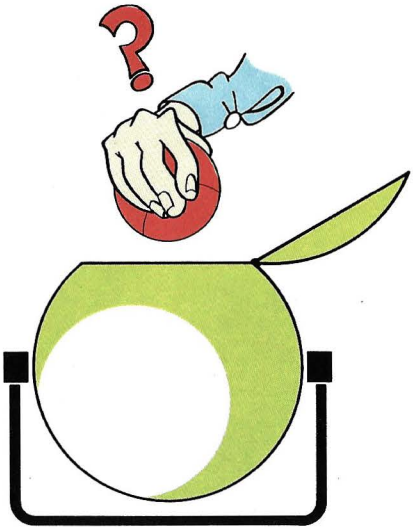
En este tipo de programas se tienen que

Números aleatorios

La mayoría de los programas de juego, sobre todo los que compiten contra el usuario, necesitan utilizar algún algoritmo que simule "espontaneidad"; esto es, que les permita no realizar siempre las mismas jugadas en situaciones idénticas. La única forma de conseguir este efecto es introduciendo una función aleatoria que genere números de forma automática e imprevisible.

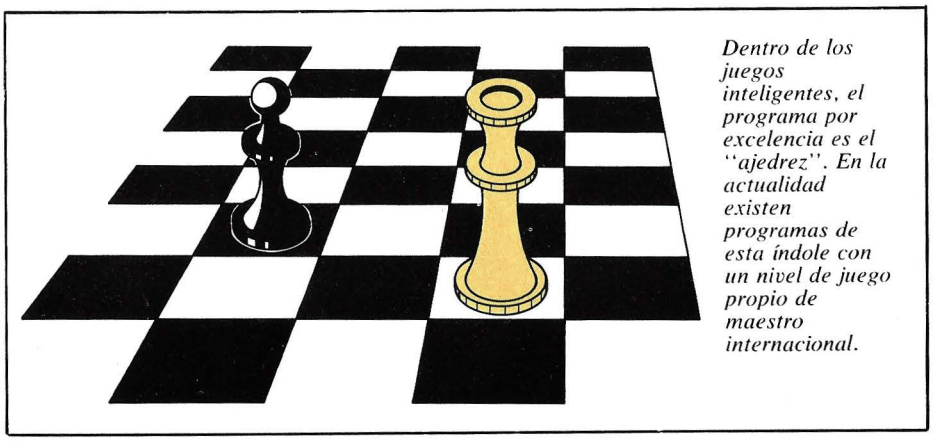
Aunque en las primeras versiones del lenguaje BASIC las funciones aleatorias debían ser generadas por el programador, en la actualidad la mayoría de los dialectos BASIC incluyen instrucciones para la generación automática de números aleatorios. Sin más que invocarlas, las instrucciones de este tipo producen de forma automática un número mayor o igual a cero y menor que la unidad.

Ello no es suficiente la mayor parte de las veces, ya que el programa necesita que el número aleatorio esté comprendido entre otros valores concretos y no entre cero y uno. Suponga que interesa obtener un número entero aleatorio (I) comprendido entre los límites A y B, ambos inclusive, y que la función aleatoria se denomina RND. Para obtenerlo, bastará con ejecutar la instrucción siguiente:




```
LET I=A+INT((B-A+1)*RND)
```

El valor aleatorio producido por la función RND se multiplica por el margen de variación admisible (B−A) incrementado en una unidad y el resultado se suma al límite inferior del intervalo (A). A partir de este punto, el número aleatorio adecuado será el representado por la variable I.



Dentro de los juegos inteligentes, el programa por excelencia es el "ajedrez". En la actualidad existen programas de esta índole con un nivel de juego propio de maestro internacional.

diferenciar dos grupos de variables, las de entrada al modelo y las de salida del mismo. Generalmente, las variables de entrada tomarán valores según la actuación del usuario. Acto seguido, el modelo las transformará mediante operaciones en variables de salida y éstas representarán en la pantalla los efectos producidos como respuesta a la entrada del usuario.
Dentro de los programas de simulación se encuentran juegos de gran espectacularidad, como los que permiten al usuario conducir un coche de fórmula-1 en reproducciones exactas de los circuitos más famosos; y también juegos de gran precisión, como los simuladores de vuelo que llegan a reproducir todos los equipos y efectos de la cabina de mando de un avión.

- *Programación pixel a pixel*

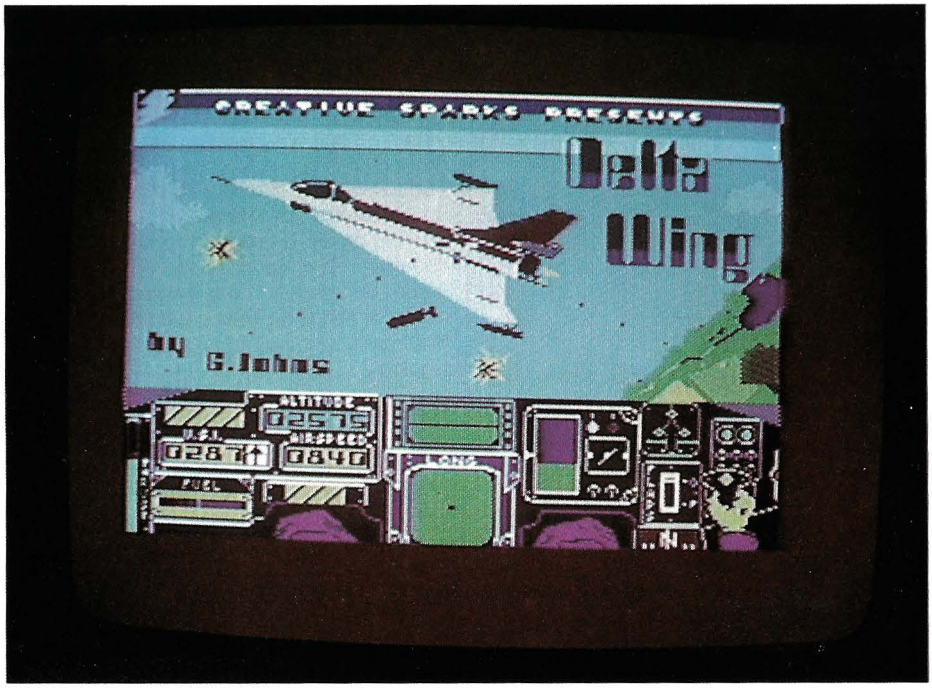
Para que sea posible aumentar la resolución de la imagen en pantalla, la mayoría de los programas de juego con animación deben trabajar independientemente con cada uno de los puntos de la pantalla: pixel a pixel (se llama pixel a cada una de las células de la superficie de visualización que puede contener blanco o negro en las pantallas monocromas, o cada uno de los posibles colores en las de color). Esta técnica se utiliza no sólo para ganar en el aspecto gráfico, sino también para incrementar la velocidad de desplazamiento de los objetos en la pantalla.

- *Utilidades del sistema*

Como último punto de apoyo de los programas de juegos y entretenimiento, cabe mencionar al propio sistema operativo del ordenador. En definitiva, serán las utilidades de cada sistema operativo las que permitirán al programador explotar al máximo la capacidad cuantitativa (tamaño de memoria, velocidad de ejecución, ...) y cualitativa (funciones, operaciones, ...) del ordenador.

PROGRAMAS INTELIGENTES

Hay que mencionar aún un último grupo de programas de entretenimiento que, por sus características técnicas, merecen ser tratados de forma aislada: los programas inteligentes.
Aunque parezca un contrasentido, se puede definir al programa inteligente como "todo aquel que es capaz de cometer errores". Aclaremos este concepto. Los programas tradicionales se limitan a procesar una serie de datos de forma exhaustiva y, cuando deben tomar una decisión, aseguran que sus resultados sean óptimos. En algunos casos, por la naturaleza exponencial de los datos tratados, resulta imposible examinar todas las posibilidades para elegir la mejor. En tal situación, el programa tiene que "razonar" qué alternativas desprecia y en cuáles profundiza. Evidentemente, según sea el razonamiento los resultados serán más o menos afortunados.
En resumen, puede establecerse el siguiente criterio de diferenciación entre programas inteligentes y convencionales: los primeros manejan conocimientos, mientras que los segundos manejan datos.
Tal vez, el juego inteligente más estudiado para su mecanización sea el ajedrez. Existen procedimientos computables que garantizan que las decisiones tomadas por el programa son óptimas. Por consiguiente, un programa que integrara estos procedimientos sería invencible; sin embargo, no existe ningún ordenador capaz de procesar este hipotético programa. La solución más verosímil para conseguir programas de ajedrez eficientes, consiste en intentar simular el comportamiento de un jugador humano y analizar tan sólo las jugadas que parezcan interesantes; de esta forma se reduce el espacio de búsqueda de jugadas y se consiguen programas satisfactorios... aunque no invencibles.



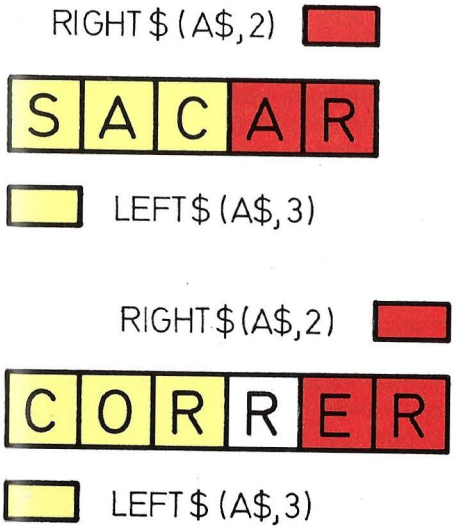
Tratamiento de cadenas

Manipulación de los datos alfanuméricos

El tratamiento de cadenas alfanuméricas confiere una particular potencia a los ordenadores al capacitarlos para manipular información no numérica. Este hecho establece una de las diferencias fundamentales entre un ordenador y una máquina de calcular programable. La actividad de las calculadoras se reduce al tratamiento de información numérica y sólo en algunos casos permiten la generación de determinados mensajes, aunque no su tratamiento.

En capítulos anteriores se han estudiado ya algunas funciones BASIC para el tratamiento de cadenas de caracteres; sin embargo, quedan aún muchas funciones en el tintero. Para proseguir con su estudio, partiremos en esta ocasión de un ejemplo "gramatical".

Suponga que a partir de un verbo, dado en forma de infinitivo, se pretende separar por un lado la terminación verbal y por otro su raíz. Esta operación no resultará difícil si se trata de un verbo cuyo número de caracteres es conocido. Por ejemplo, imagine el verbo "sacar":



Actuación de las funciones LEFT$ y RIGHT$ sobre una cadena de caracteres. En el primer caso, se extrae correctamente la raíz y la terminación verbal de la palabra operada, mientras que en el segundo (correr), no se obtiene la raíz verbal en su integridad.

```
10 PRINT RIGHT$("SACAR",2)
20 PRINT LEFT$("SACAR",3)
30 END
■
```

El resultado de este simple programa es el que aparece en la pantalla:

```
RUN
AR
SAC
■
```

No se ha presentado problema alguno ya que, en este caso, el verbo a tratar era conocido de antemano. No obstante, el programa puede complicarse algo más, en orden a permitir la introducción de cualquier verbo:

```
10 INPUT A$
20 PRINT RIGHT$(A$,2)
30 PRINT LEFT$(A$,3)
40 STOP
■
```

Ahora, el dato (verbo) no es fijo, sino que será leído por efecto de la instrucción INPUT que aparece en la línea 10.

```
RUN
?SACAR <CR>
AR
SAC
■
```

¿Qué ocurrirá si el dato introducido es un verbo cuyo infinitivo tiene más o menos de cinco caracteres? Por ejemplo, "correr":

```
RUN
?CORRER <CR>
ER
COR
■
```

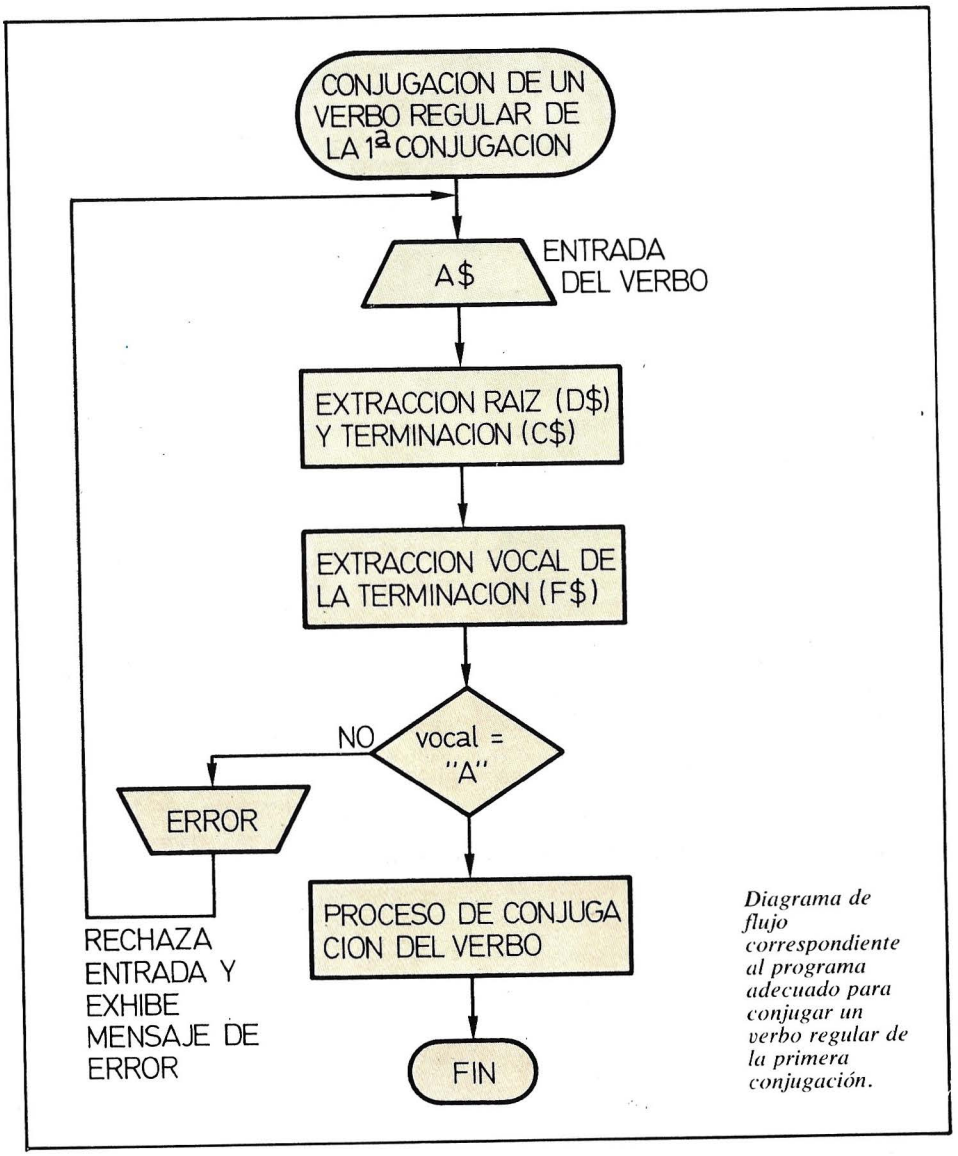


Diagrama de flujo correspondiente al programa adecuado para conjugar un verbo regular de la primera conjugación.

El resultado no es correcto, falta una letra en la zona de "raíz". Es evidente que para que el programa resulte eficaz éste debe ser capaz de medir la longitud del dato de entrada (el verbo) y fraccionarlo en consecuencia.

El lenguaje BASIC dispone de una función que permite evaluar la longitud de una cadena de caracteres. Esta es LEN, cuyo formato general coincide con:

LEN (<cadena>)

LEN

Formato:
LEN(<cadena>)
Donde <cadena> es la cadena sobre la que ha de operar la función.
Función:
Permite calcular la longitud de una cadena.
Ejemplos:
LEN(A$)
LEN("CASA")

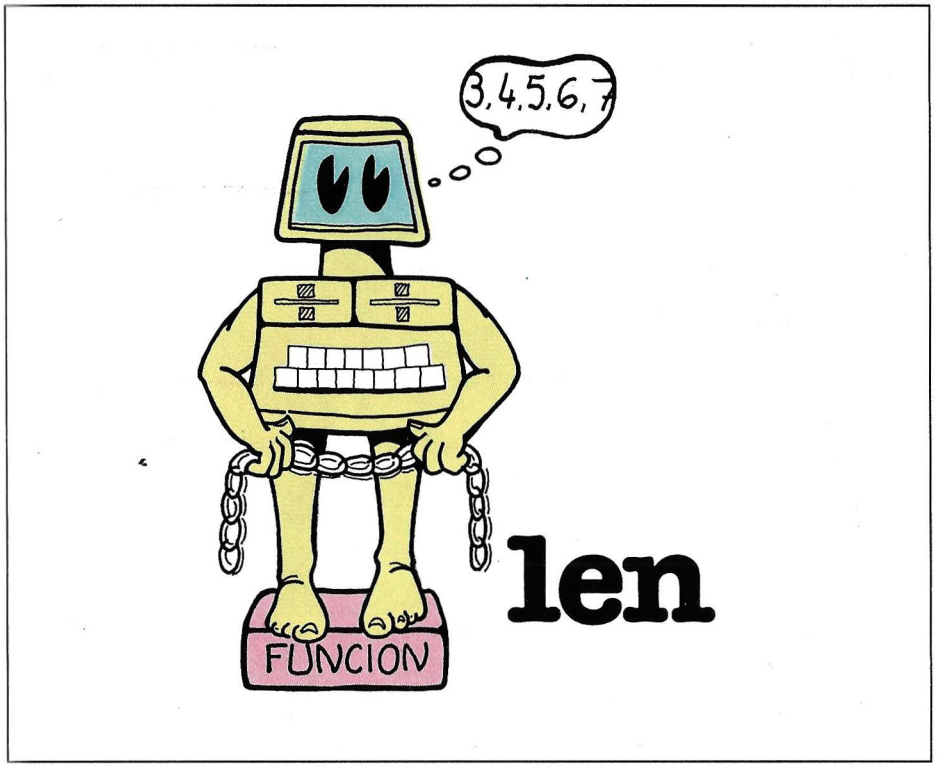
Esta función da como resultado un número natural, que coincide con el número de caracteres que constituyen la cadena especificada en el argumento (incluyendo los espacios en blanco).
Hay que recordar que LEN es una función y, por lo tanto, será preciso utilizarla asociada a un comando.
Veamos cuál será el nuevo aspecto del programa ejemplo, una vez introducida la función LEN:

```
10 INPUT A$
20 LET B=LEN(A$)
30 LET C$=RIGHT$(A$,2)
40 LET D$=LEFT$(A$,B-2)
50 PRINT C$
60 PRINT D$
70 END
```

La ejecución del programa conduce al siguiente resultado:

```
RUN
?CORRER <CR>
ER
CORR
■
```

En efecto, el problema ha dejado de existir. El programa admite ahora cualquier verbo, sea cual fuere su longitud o número de caracteres. La razón de ello estriba en que la terminación verbal siempre coincide con las dos últimas letras (RIGHT$(A$, 2). Por lo tanto, la raíz ha de estar compuesta por las restantes letras. cuyo número será igual a B−2: la longitud total calculada por medio de la función LEN, menos las dos letras de la terminación verbal.
El uso de la referida función puede permitir, incluso, la confección de un programa capaz de conjugar un verbo. Sencillamente, añadiendo a la raíz las terminaciones de las distintas personas según el tiempo verbal elegido. Desde luego, habrá que olvidar a los verbos irregulares. Suponga que se desea que el ordenador conjugue un verbo de la primera conjugación, cuyas terminaciones del presente de indicativo son:



— o
— as
— a
— amos
— ais
— an

Con un sencillo programa BASIC puede instruirse al ordenador para que sea él quien se encargue de extraer la raíz del verbo y, acto seguido, una a ésta la correspondiente terminación y el pronombre adecuado.

```
 10 INPUT A$
 20 LET B=LEN(A$)
 30 LET B$=LEFT$(A$,B-2)
100 PRINT "YO "+B$+"O"
110 PRINT "TU "+B$+"AS"
120 PRINT "EL "+B$+"A"
130 PRINT "NOSOTROS "+B$+"AMOS"
140 PRINT "VOSOTROS "+B$+"AIS"
150 PRINT "ELLOS "+B$+"AN"
160 END
```

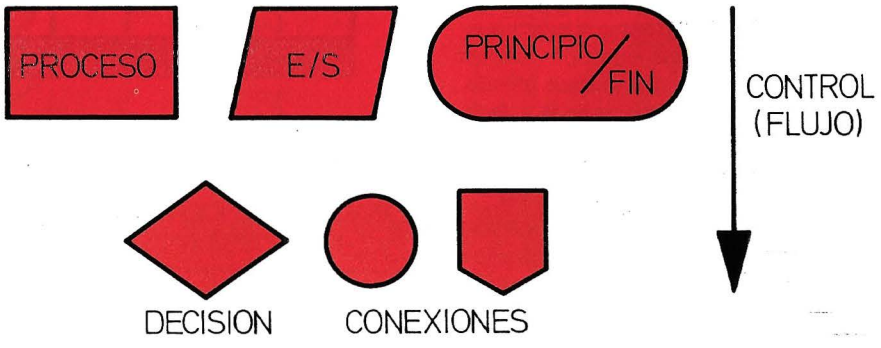
La primera zona del programa (instrucciones 10, 20, 30) evalúan la longitud de la

Diagramas de flujo

La fase previa a la codificación de un programa se concreta en el análisis y descomposición del problema a resolver. Ello empieza con el estudio del algoritmo o método de solución, y más tarde, mediante la descomposición del algoritmo en una serie de acciones básicas reflejadas en un ordinograma.
Los diagramas de flujo, también llamados flujogramas u ordinogramas, son gráficos que permiten la representación simbólica del algoritmo. Con ello se consigue ordenar un problema en una sucesión de pasos que permitirán desglosarlo. Por lo tanto, estos flujogramas facilitan la posterior codificación e introducción en la máquina del algoritmo para resolver el problema.
Cuando un algoritmo ha sido traducido a ordinograma, es más fácil codificar el programa utilizando un lenguaje informático.
El orden lógico para confeccionar un programa destinado al ordenador se concreta en los siguientes puntos:

1. Estudio del algoritmo.
2. Traducción del algoritmo a ordinograma.
3. Confección del programa en el lenguaje apropiado.

Muchos programadores piensan que no es necesario pasar por la fase de construir el ordinograma. Sin embargo, la experiencia dicta que, en el 90 % de los casos, tal omisión sólo contribuye a dificultar la escritura del programa. Hasta llegar incluso al extremo de imposibilitar su eficaz puesta a punto.
Los ordinogramas se componen de dos zonas fundamentales. Por un lado las órdenes o acciones que debe ejecutar el ordenador; éstas se enmarcan dentro de una serie de figuras geométricas. Por otro, la secuencia de control o de procesado de órdenes, que se representa por medio de líneas que enlazan a las diversas acciones.
Cada tipo de orden o acción se enmarca dentro de una determinada figura geométrica. Las principales son las que ilustra la figura adjunta.
Es importante observar que el ordinograma resulta de fácil lectura y constituye una documentación del programa eficaz e inteligible. En él pueden utilizarse frases del lenguaje corriente, abreviaturas, e incluso instrucciones próximas a los lenguajes de programación.



Repertorio de símbolos básicos para la confección de ordinogramas.

palabra introducida (el verbo a conjugar) y depositan la raíz en la variable B$. A continuación entra en escena el segundo bloque de instrucciones (a partir de la línea 100) que se ocupará de conjugar el presente de indicativo de un verbo regular de la primera conjugación.
Por ejemplo:

```
RUN
?AMAR <CR>
YO AMO
TU AMAS
EL AMA
NOSOTROS AMAMOS
VOSOTROS AMAIS
ELLOS AMAN
■
```

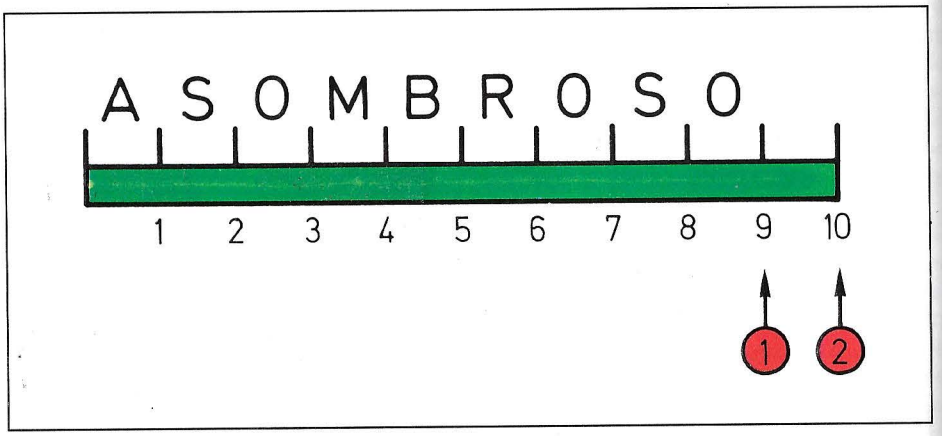
El programa puede ampliarse para que sea capaz de conjugar más tiempos verbales. Veamos, por ejemplo, qué habrá que hacer con un tiempo compuesto: el pretérito perfecto.
La forma general o "recetilla" para generarlo es la siguiente: al pretérito imperfecto del verbo haber, hay que añadirle el participio del verbo que se desea conjugar. De manera que, esquemáticamente, puede representarse de la siguiente forma:

```
YO HE-ADO
TU HAS-ADO
EL HA-ADO
NOSOTROS HEMOS-ADO
VOSOTROS HABEIS-ADO
ELLOS HAN-ADO
```

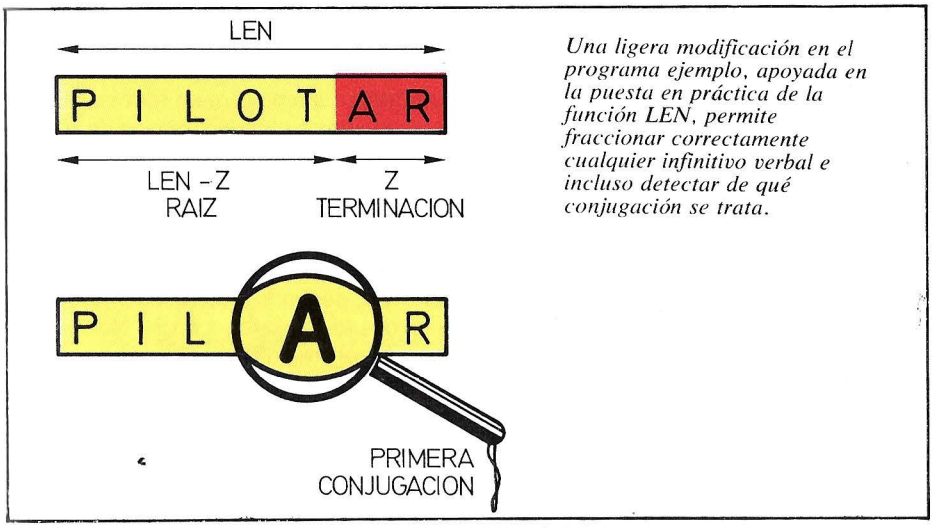
La raíz del verbo, extraída con las líneas 10, 20 y 30 del programa, hay que unirla a la terminación indicada tras el guión. De esta forma es posible crear una rutina que permita al ordenador conjugar dicho tiempo verbal. Siempre y cuando, desde luego, se trate de un verbo regular y de la primera conjugación:

```
 10 INPUT A$
 20 LET B=LEN(A$)
 30 LET B$=LEFT$(A$,B-2)
200 PRINT "YO HE "+B$+"ADO"
210 PRINT "TU HAS "+B$+"ADO"
220 PRINT "EL HA "+B$+"ADO"
230 PRINT "NOSOTROS
    HEMOS "+B$+"ADO"
```

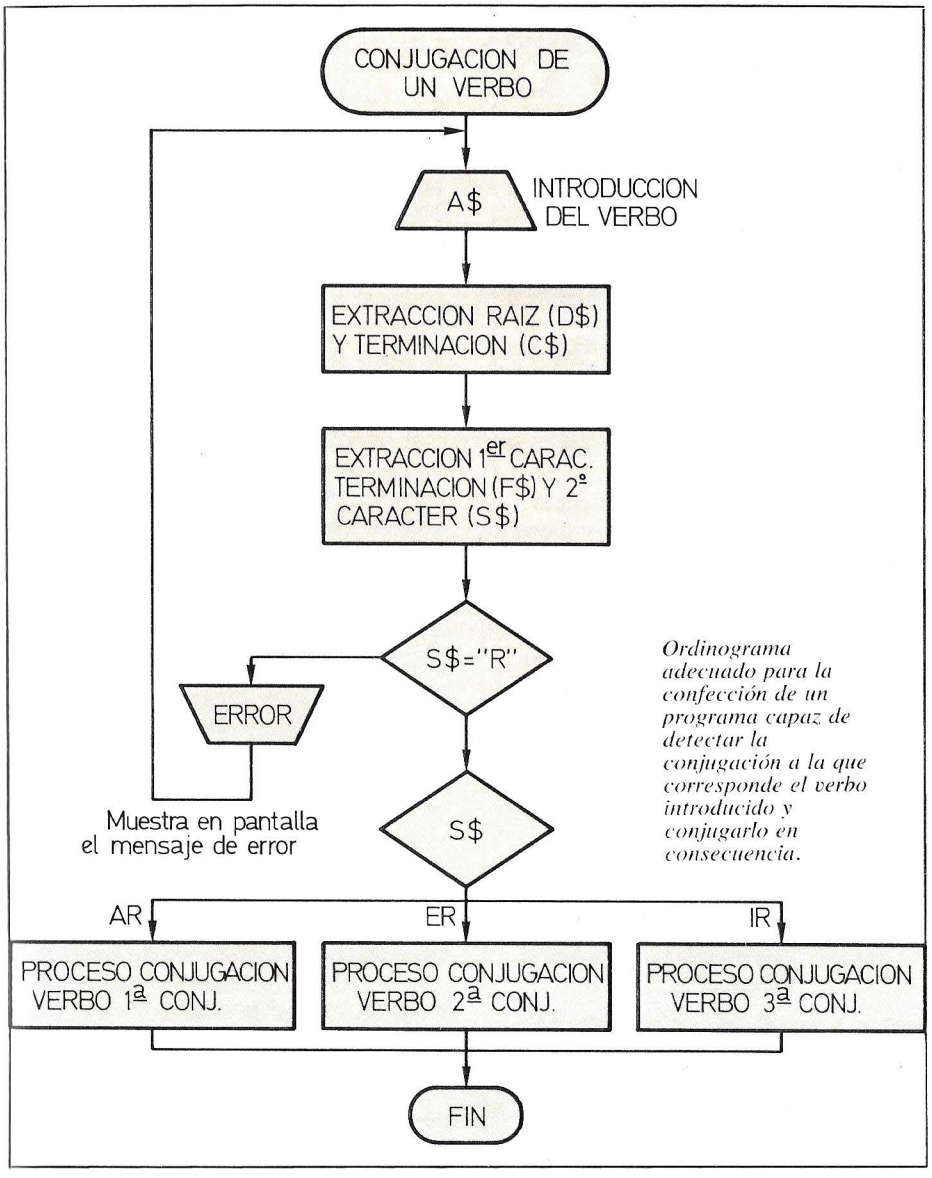
Internamente, el ordenador manipula los caracteres a través de su representación en código ASCII. El lenguaje BASIC incorpora dos herramientas adecuadas para realizar su traducción desde el programa: estas son las funciones ASC y CHR$.



La función LEN permite medir la longitud de una cadena alfanumérica, incluyendo los espacios en blanco. En el caso de la figura, si la cadena coincidiera con la palabra "asombroso" su longitud sería 9, mientras que si se incluye un espacio en blanco al final de la misma ("asombroso") su longitud será de 10 caracteres.



Una ligera modificación en el programa ejemplo, apoyada en la puesta en práctica de la función LEN, permite fraccionar correctamente cualquier infinitivo verbal e incluso detectar de qué conjugación se trata.



Ordinograma adecuado para la confección de un programa capaz de detectar la conjugación a la que corresponde el verbo introducido y conjugarlo en consecuencia.

```
240 PRINT "VOSOTROS
    HABEIS "+B$+"ADO"
250 PRINT "ELLOS HAN "+B$+"ADO"
```

Cabe observar que en esta rutina se repite en todas las líneas de la cadena la zona B$+"ADO". Desde luego, tal reiteración puede subsanarse para facilitar al usuario la introducción del programa. El método consiste en otorgarle un nombre y realizar la asignación bajo ese nombre. De ello se ocupará la instrucción siguiente:

```
90 P$=B$+"ADO"
```

De esta forma, la variable P$ almacenará el participio simple del verbo a conjugar (raíz+ADO). Ello supone modificar las líneas 200 a la 250, para que su nuevo aspecto coincida con:

```
PRINT "<pronombre> <verbo
   haber>"+P$
```

Aplicando este mismo procedimiento, es posible programar la conjugación de los restantes tiempos verbales regulares de la primera conjugación, e incluso de las restantes. Sin embargo, en principio, no se puede hacer gran cosa con los verbos irregulares.

Para dar por terminado con el ejemplo que nos ocupa, tal vez sea interesante apuntar cómo habrá que instruir al ordenador para que aprenda a diferenciar entre una y otra conjugación. La diferencia, como de todos es sabido, consiste en la vocal de la terminación verbal (ar, er, ir). De modo que la rutina necesaria para comprobar si un determinado verbo es de la primera conjugación puede coincidir con la siguiente:

```
10 INPUT A$
20 LET B=LEN(A$)
30 LET C$=RIGHT$(A$,2)
50 LET F$=LEFT$(C$,1)
60 IF F$< >"A"THEN PRINT
   "ERROR":GOTO 10
```

Como se observa, la variable F$ almacena la vocal que permitirá distinguir de qué conjugación es el verbo introducido. La línea 60 se encarga de rechazar los verbos que no pertenezcan a la primera conjugación y, a su vez, devuelve la secuencia de ejecución al principio del programa.

El programa que sigue resulta adecuado para conjugar los tiempos de presente de indicativo y pretérito perfecto de los verbos regulares de la primera conjugación (infinitivo terminado en "ar").

CHR$

Formato:
CHR$(<número>)
<número>. Es un número natural que no ha de ser mayor de 255.
Función:
Produce una cadena de un único carácter cuyo código ASCII corresponde al <número> especificado.
Ejemplos:
CHR$(32)
CHR$(7)

```
10 INPUT A$
20 LET B=LEN(A$)
30 LET C$=RIGHT$(A$,2)
40 LET D$=LEFT$(A$,B-2)
50 LEF F$=LEFT$(C$,1)
60 IF F$< >"A" THEN PRINT
   "ERROR":GOTO 10
90 P$=D$+"ADO"
95 PRINT "PRESENTE DE INDICATIVO"
100 PRINT "YO "+D$+"O"
110 PRINT "TU "+D$+"AS"
120 PRINT "EL "+D$+"A"
130 PRINT "NOSOTROS "+D$+"AMOS"
140 PRINT "VOSOTROS "+D$+"AIS"
150 PRINT "ELLOS "+D$+"AN"
180 PRINT "PRETERITO PERFECTO"
200 PRINT "YO HE "+P$
210 PRINT "TU HAS "+P$
220 PRINT "EL HA "+P$
230 PRINT "NOSOTROS HEMOS "+P$
240 PRINT "VOSOTROS HABEIS "+P$
250 PRINT "ELLOS HAN "+P$
```

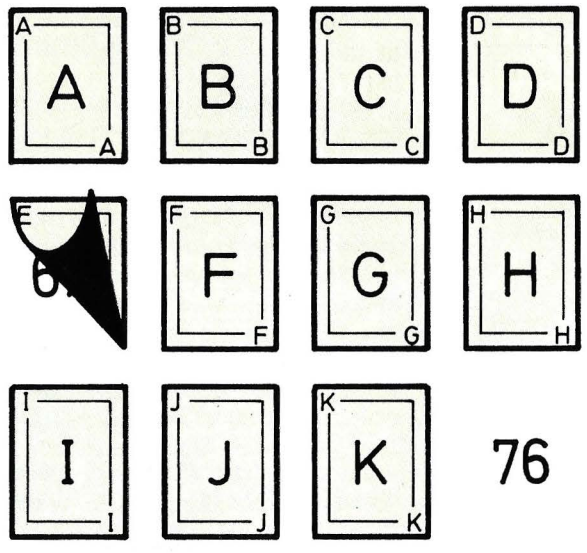
El programa es capaz de detectar si se ha introducido un verbo no perteneciente a la primera conjugación mostrando, en tal caso, el mensaje "ERROR" y solicitando de nuevo el infinitivo del verbo.
Para detectar si la palabra introducida es un verbo o no, únicamente cabría analizar si la segunda letra de la terminación es una erre. A la hora de proceder a esta comparación será necesario efectuar las siguientes modificaciones en el programa:
— Incluir una línea de número 55 que extraiga el carácter final de la palabra, por ejemplo:

```
55 LET S$=RIGHT$(C$,1)
```

— Modificar la comparación de la línea 60 para que sea capaz de detectar si S$ es o no una erre.
El resultado de estas modificaciones queda plasmado en la siguiente línea de programa:

```
60 IF F$< >"A" OR S$< > "R" THEN
   PRINT "ERROR":GOTO 10
```

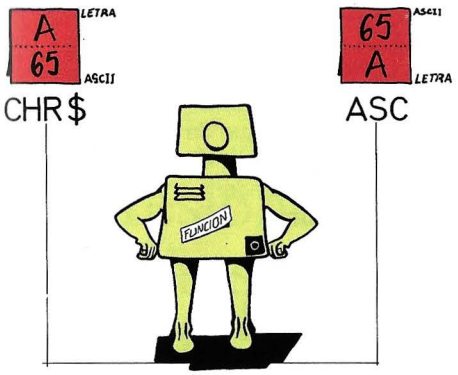
Tal como se ha indicado, el programa es capaz de conjugar los tiempos presente de indicativo y pretérito perfecto y, a su vez, permite distinguir si se ha introducido un verbo de una conjugación distinta de la primera o una palabra no terminada en R, con lo cual se evita la aparición de algunos errores a la hora de introducir el verbo. En el gráfico adjunto se observa el diagrama de flujo correspondiente a este programa. También es posible construir un programa capaz de distinguir entre un verbo de la primera, segunda o tercera conjugación, y conjugarlo en consecuencia. No vamos a escribir el programa en cuestión, si bien la figura ilustra el ordinograma necesario para realizarlo.



El ordenador utiliza el código ASCII para representar internamente a los caracteres. Por medio de la función ASC es posible obtener el código ASCII que corresponde a cada carácter.

CODIGO ASCII

Las cadenas de caracteres se almacenan en el ordenador en código ASCII. Como ya es sabido, el código ASCII es un método de representación binario adecuado para codificar a los diversos caracteres alfanuméricos y símbolos especiales. El código de cada carácter ha de almacenarse en ocho bits, de ahí que sólo pueda variar entre cero y 255 en su equivalencia decimal (2^8=256 posibles representaciones). Sin embargo, no suelen utilizarse todos los posibles códigos.
Los restantes códigos se reservan para otros cometidos. Por ejemplo, para la representación de caracteres gráficos especiales y caracteres de control. Mediante éstos se canalizan órdenes destinadas al propio ordenador y a los dispositivos periféricos asociados al mismo.
El código ASCII es compartido por distintos ordenadores por lo que respecta a la representación de los caracteres alfabéticos, numéricos y algunos signos de puntuación; no obstante, suele diferir en lo relativo a los caracteres de control. Aunque bien es cierto que algunos códigos pueden coincidir, tal es el caso del retroceso de carro (13), avance de línea (10) y ESCAPE (27). De todas formas, lo mejor es consultar siempre la tabla de códigos de cada equipo.
El lenguaje BASIC permite realizar ciertas manipulaciones con los códigos ASCII. Para empezar, es posible definir una cadena constituida por un carácter expre-



sado por medio de su código ASCII. La función adecuada es CHR$( ), cuyo formato genérico adopta el siguiente aspecto:

CHR$(<número>)

coincidiendo el argumento <número> con el código ASCII del carácter deseado. Por ejemplo:

```
PRINT CHR$(48)
B$=CHR$(68)
PRINT CHR$(A)
LET B$=CHR$(A+2)
```

También es posible encadenar varias de estas funciones para construir una cadena más larga; por ejemplo:

```
PRINT CHR$(49)+CHR$(50)+CHR$(49)
<CR>121

PRINT "AS"+CHR$(65) <CR>
ASA
■
```

Otra función BASIC de esta categoría es ASC, adecuada para obtener el código ASCII del primer carácter de una cadena dada. Su formulación es la siguiente:

ASC (<cadena>)

El argumento <cadena> debe contener a la cadena alfanumérica afectada. La respuesta a la función ASC coincidirá con el código ASCII correspondiente a su primer carácter, empezando por la izquierda. Por ejemplo:

TABLA DE CONVERSION

ORDENADOR	LEN	CHR$	ASC
	LEN (<c>)	CHR$ (<n>)	ASC (<c>)
APPLE II (APPLESOFT)	LEN (<c>)	CHR$ (<n>)	ASC (<c>)
APRICOT (M-BASIC)	LEN (<c>)	CHR$ (<n>)	ASC (<c>)
ATARI	LEN (<c>)	CHR$ (<n>)	ASC (<c>)
CBM 64	LEN (<c>)	CHR$ (<n>)	ASC (<c>)
DRAGON	LEN (<c>)	CHR$ (<n>)	ASC (<c>)
EQUIPOS MSX	LEN (<c>)	CHR$ (<n>)	ASC (<c>)
HP-150	LEN (<c>)	CHR$ (<n>)	ASC (<c>)
IBM PC	LEN (<c>)	CHR$ (<n>)	ASC (<c>)
MPF	LEN (<c>)	CHR$ (<n>)	ASC (<c>)
NCR DM-V (MS-BASIC)	LEN (<c>)	CHR$ (<n>)	ASC (<c>)
NEW BRAIN	LEN (<c>)	CHR$ (<n>)	ASC (<c>)
ORIC	LEN (<c>)	CHR$ (<n>)	ASC (<c>)
SHARP MZ-700 (MZ-BASIC)	LEN (<c>)	CHR$ (<n>)	ASC (<c>)
SINCLAIR QL	LEN (<c>)	CHR$ (<n>)	CODE (<c>)
SPECTRAVIDEO	LEN (<c>)	CHR$ (<n>)	—
ZX-SPECTRUM	LEN (<c>)	CHR$ (<n>)	CODE (<c>)

<c>: Cadena de caracteres o variable alfanumérica. <n>: Número coincidente con el código ASCII de un carácter.

```
PRINT ASC("A")
PRINT ASC("ALTA")
PRINT ASC(B$)
A=ASC(L$)
```

A partir de las funciones definidas, puede confeccionarse un programa capaz de operar la conversión de un número binario a decimal.

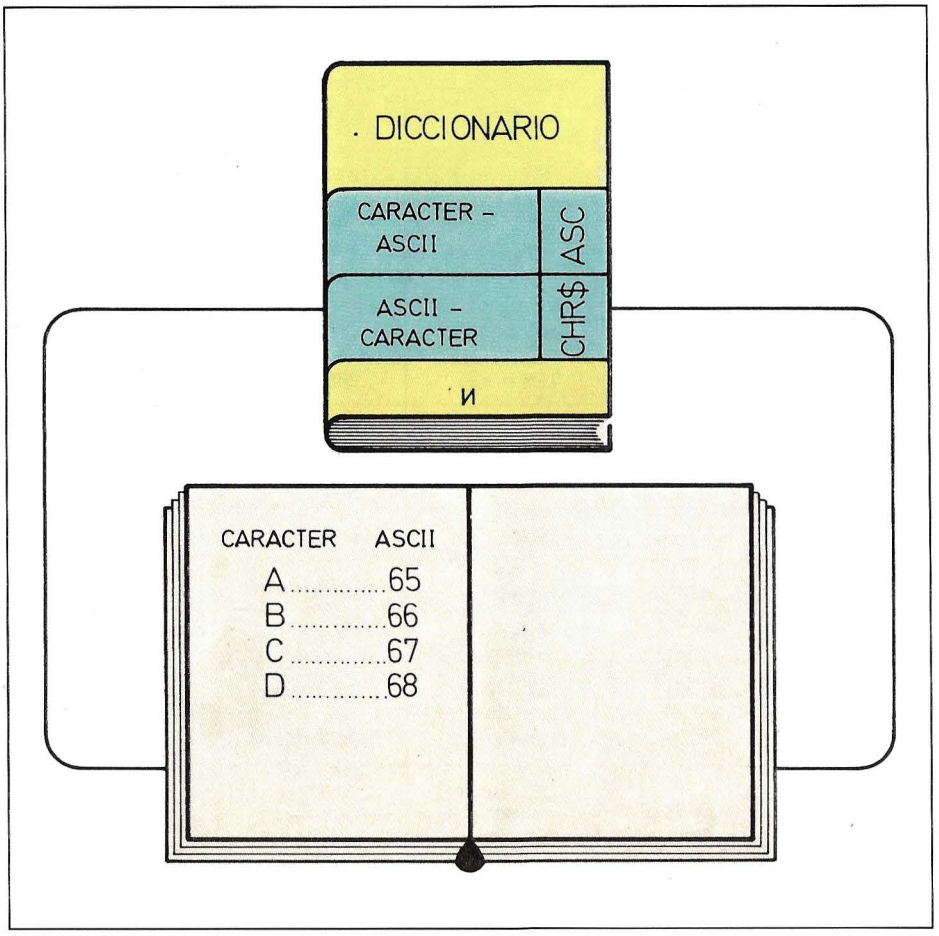
En primer lugar es oportuno conocer la "receta" para efectuar la conversión de binario a decimal. Para empezar hay que tomar cada uno de los dígitos del número binario, a partir del situado más a la dere-

ASC

Formato:
ASC(<cadena>)
<cadena> es la cadena de caracteres sobre la que opera la función.

Función:
Esta función da como resultado un número que corresponde al código ASCII del primer carácter de la cadena sobre la que opera.

Ejemplos:
ASC(A$)
ASC("S")



ASC y CHR$ son las funciones BASIC especializadas en la traducción de caracteres a código ASCII y viceversa.

cha, y multiplicarlo por la potencia de dos que corresponda al orden que ocupa, empezando por la potencia cero. El valor de cada una de las respectivas potencias de 2 es lo que se llama *peso* del dígito en cuestión.
Así, por ejemplo, el número binario 10110101 presenta la siguiente correspondencia entre bits y pesos:

```
1   0  1  1  0  1  0  1
2⁷  2⁶ 2⁵ 2⁴ 2³ 2² 2¹ 2⁰
128 64 32 16 8  4  2  1
```

Su homólogo en el sistema de numeración decimal se obtiene, sencillamente, sumando los productos de cada dígito por su peso respectivo. Esto es:

$1\times2^0+0\times2^1+1+2^2+0\times2^3+1\times2^4+1\times2^5+$
$+0\times2^6+1\times2^7=1\times1+0+1\times4+0+1\times16+$
$+1\times 32+0+1\times128=1+4+16+32+128=181$

En definitiva, el número binario 10110101 equivale al 181 en el sistema de numeración decimal.
Y pasemos ahora al programa adecuado para operar esta conversión de forma automática.
En primer lugar, será preciso introducir la cadena de ceros y unos y evaluar su longitud. De ello se ocuparán las siguientes instrucciones:

```
10 INPUT "EL NUMERO BINARIO A
   CONVERTIR ES:";A$
20 B=LEN(A$):REM NUMERO DE DIGITOS
   DE LA CADENA
```

Tras ello es necesario extraer cada uno de los dígitos o bits de la cadena y comprobar si es cero o uno. Por supuesto, si el dígito no coincide con 0 ó 1, habrá que mostrar un mensaje de error. A continuación, hay que multiplicar cada dígito por su peso correspondiente.

Todo ello correrá a cargo de un bucle. Este extraerá los caracteres, empezando por el situado más a la izquierda; en consecuencia, el peso del primer carácter será dos elevado a la longitud de la cadena menos uno. El siguiente será dos elevado a la longitud de la cadena menos dos y así sucesivamente.

```
30 FOR I=1 TO B
40 B$=MID$(A$,I,1)
50 C=ASC(B$)
60 IF C<48 OR C<49 THEN PRINT
   "ERROR":GOTO 10
70 LET C=C-48
80 LET N=C*2↑(B-I)+N
90 NEXT I
100 PRINT "LA SOLUCION ES:";N
110 END
```

La instrucción de la línea 70 convierte la cadena "0" o la cadena "1" en el número 0 ó 1 según el caso. Para entender su actuación hay que recordar que el código ASCII del "0" es 48 y el del "1" es 49. En consecuencia, al restar el valor 48 del código en cuestión, se obtendrá el número correspondiente.
Una vez agrupadas las distintas zonas, el programa conjunto ofrecerá el siguiente aspecto:

```
 5 LET N=0
10 INPUT "EL NUMERO BINARIO A
   CONVERTIR ES:";A$
20 B=LEN(A$): REM NUMERO DE DIGITOS
   DE LA CADENA
30 FOR I=1 TO B
40 B$=MID$(A$,I,1)
50 C=ASC(B$)
60 IF C<48 OR C<49 THEN PRINT
   "ERROR":GOTO 5
70 LET C=C-48
80 LET N=C*2↑(B-I)+N
90 NEXT I
100 PRINT "LA SOLUCION ES:";N
110 END
```

Desde luego, el programa es susceptible de incorporar modificaciones a voluntad del usuario. Incluso el método adoptado podría haber sido otro. El hecho de optar por este procedimiento de conversión se debe a que es posible operar conversiones de números expresados en una base distinta de la binaria; por ejemplo, hexadecimal u octal. De ahí que la adaptación del programa para que sea capaz de realizar otro tipo de conversión resulte de lo más inmediato.

Pascal (2)

Expresiones generales y tipos escalares de datos

En PASCAL existen cuatro tipos normalizados de datos; enteros, reales, caracteres y booleanos. Estos tipos están incluidos en todo compilador de PASCAL; pero como se verá más adelante, el PASCAL permite definir al usuario sus propios tipos de datos. La notación "escalar" se refiere a que son datos de una única dimensión, o lo que es lo mismo: que sólo definen una característica del elemento al que se refieren. No hay que olvidar que el PASCAL permite además la estructuración de los datos en formas más complejas. Cada tipo de datos tiene su manera particular de expresar las constantes, así como sus propias operaciones y operadores definidos que permiten obtener valores de un tipo determinado con operandos de ese mismo tipo.

DATOS DE TIPO ENTERO

Son los que presentan a un número entero, esto es: a una secuencia de dígitos decimales precedidos o no por un signo aritmético de suma o resta. En una constante entera no se admiten espacios en blanco, comas, ni puntos.

El delimitador o palabra clave que permite declarar datos de tipo entero es INTEGER.

Ejemplo:

VAR DIAS, MESES: INTEGER;

Cuando se desee evaluar una expresión en la que los operandos son enteros, los siguientes operadores producirán un resultado de tipo entero:

— Suma (+): a+b; suma a y b.

— Resta (–): a–b; resta de a el valor de b.

— Multiplicación (*): a*b; multiplica a * b.

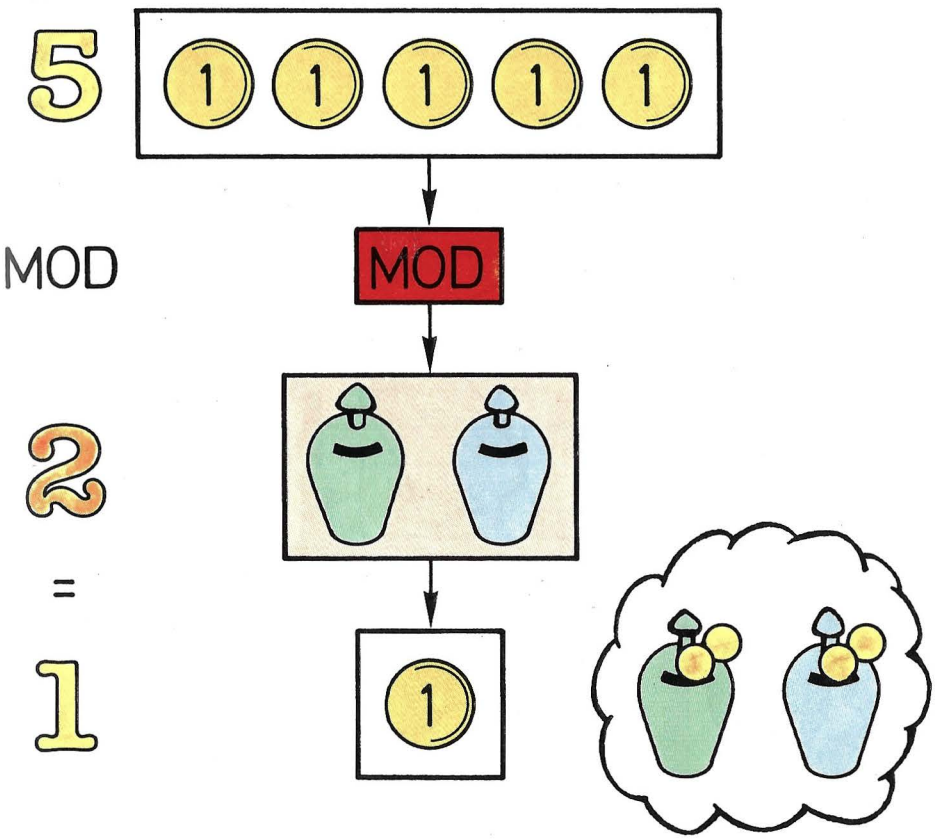
— División (DIV): a DIV b; divide a entre b y trunca los decimales.

— Resto entero (MOD): a MOD b; calcula el resto entero de la división de a entre b.

Ejemplos:

16 DIV 4 (el resultado es 4)

4 DIV 3 (el resultado es 1)



El operador MOD obtiene el resto entero de una división entre dos números enteros.

DATOS DE TIPO REAL

El tipo real define a un elemento del conjunto de los números reales. Estos datos se representan con una parte entera y una parte fraccionaria, separadas por un punto decimal y precedidos o no por el signo correspondiente.

Cuando se quieran representar números reales muy grandes o muy pequeños, es conveniente utilizar la notación científica o exponencial. Esta incluye una "E" seguida por el exponente de la potencia de 10 que corresponda. Por ejemplo, para represen-

tar el número un millón se utilizará: 1.0 E 6, ya que no se puede omitir la parte entera ni la parte fraccionaria. Por último, cabe señalar que las variables de tipo real se declaran con la palabra REAL.
Por ejemplo:
VAR TEMPERATURA: REAL;
Si se tiene una expresión en la que al menos uno de los operandos es de tipo real (el otro puede ser de tipo entero), los siguientes operadores obtendrán un valor de tipo real como resultado:
— Suma (+): a+b; suma a con b.
— Resta (−): a−b; resta de a el valor de b.
— Multiplicación (*): a*b; multiplica a por b.
— División (/): a/b; divide a entre b.
En el caso de la división, el resultado será de tipo real aunque los operandos sean de tipo entero.

DATOS DE TIPO CARACTER

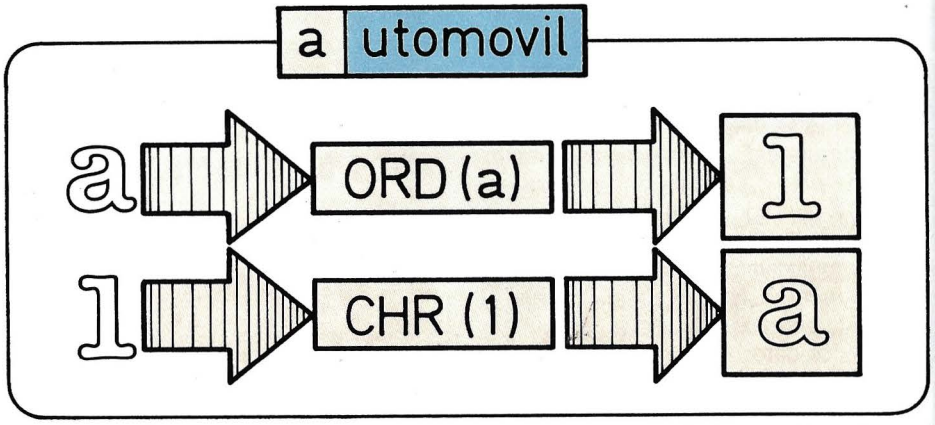
Este tipo de datos consiste en un carácter encerrado entre apóstrofes (' '). Por ejemplo: 'A', '2', '*'... son datos de tipo carácter. Para representar un apóstrofe, éste se escribe dos veces: ('''').
Cada variable de tipo carácter tomará el valor de un solo carácter y su declaración en el programa se hace por medio de la palabra CHAR.
Ejemplo:
VAR LETRA: CHAR;
Para este tipo de datos no existen operadores específicos que permitan efectuar cálculos con ellos; no obstante, sí se pueden introducir las constantes y las variables de tipo carácter de lectura, escritura y asignación.
En contrapartida, el PASCAL tiene definidas dos funciones estandarizadas que sólo son aplicables a los datos de tipo carácter. Estas son:
— ORD(<car.>) que proporciona el número ordinal correspondiente al carácter <car.>, y
— CHR(<num.>) que da como resultado el carácter cuyo número ordinal es <num.>.
Ambas funciones suelen también denominarse de transferencia, por permitir la identificación entre cada carácter y su número ordinal correspondiente.

DATOS DE TIPO BOOLEANO

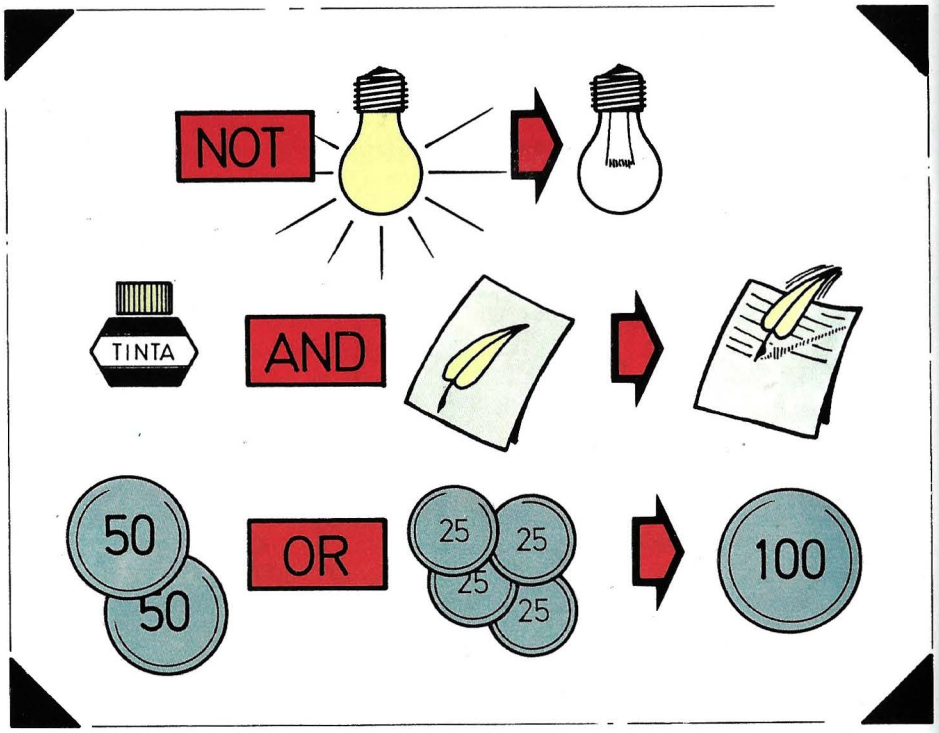
Un dato se dice que es BOOLEANO, o que pertenece al álgebra de Boole, cuando sólo puede tomar dos valores: abierto o cerrado, alto o bajo, cierto o falso, uno o cero...
En PASCAL existen dos palabras reservadas para definir a cada uno de estos valores: TRUE y FALSE (cierto o falso).

Para entenderlo mejor, cabe pensar en qué situación está una ventana: puede estar abierta o cerrada, aunque no es admisible que esté a la vez abierta y cerrada, y tampoco puede estar de otra forma que no sea alguna de ambas. La situación de la ventana será por tanto una variable *booleana*. A una de las situaciones se la asigna el valor TRUE y a la otra FALSE.
Para declarar un dato de tipo Booleano se utiliza la palabra BOOLEAN.
Por ejemplo:

VAR CUENTO: BOOLEAN;



Las funciones ORD y CHR que forman parte del lenguaje PASCAL relacionan cada carácter con la posición que éste ocupa dentro de una palabra.



Los operadores lógicos NOT, AND y OR manejan variables de tipo booleano; estas sólo pueden adoptar dos valores: TRUE (cierto) o FALSE (falso).

Entre operandos de tipo booleano se pueden aplicar los siguientes operadores que proporcionan un valor de tipo booleano:
— NOT obtiene el valor opuesto al del resto del dato booleano sobre el que se aplica. Realiza la operación lógica "negación".
— AND efectúa el producto lógico; el resultado será FALSE, si al menos uno de los operandos tiene el valor FALSE.
— OR representa la operación de suma lógica; en ella, el resultado será TRUE si al menos uno de los operandos tiene el valor TRUE.

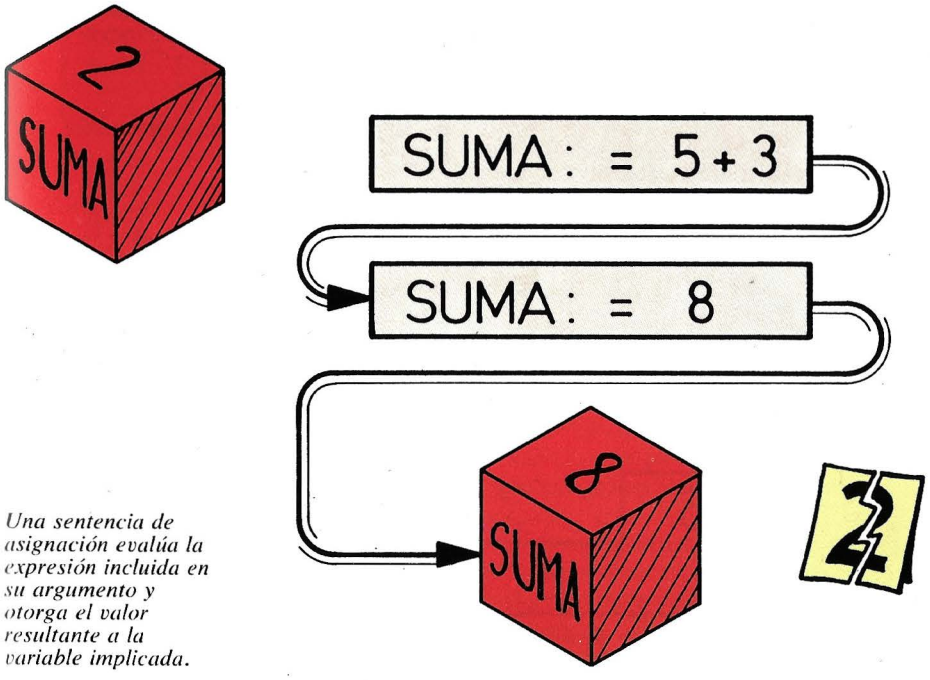
Existen además una serie de operadores que se denominan *relacionales,* éstos obtienen un valor booleano aunque los operandos no sean de tipo booleano:

= : indica la igualdad entre dos expresiones
< > : operador de desigualdad
< : relación "menor que"
> : relación "mayor que"
<= : relación "menor o igual que"
>= : relación "mayor o igual que"

Ejemplos de expresiones booleanas pueden ser:

A < > 'B'
N>=0
A AND (NOT B)

El contenido de una variable puede ser borrado y actualizado con un nuevo valor por medio de sentencias de asignación.



Una sentencia de asignación evalúa la expresión incluida en su argumento y otorga el valor resultante a la variable implicada.

LA SENTENCIA DE ASIGNACION

Una vez vistos los fundamentos básicos del PASCAL, y estudiado el concepto de variable y su declaración de tipo, hay que analizar la forma en la que pueden utilizarse las variables dentro del contexto del programa.
Cada variable se puede imaginar como una hoja en blanco de un block de notas, en la que se puede ir "anotando" con lápiz un valor constante; éste coincidirá con el valor actual que contiene dicha variable. Posteriormente, es posible "borrar" el contenido de esa hoja del block y volver a anotar otro valor, con lo que se habrá actualizado el contenido de la variable en cuestión.
Estas dos acciones tan sencillas, pero a la vez tan importantes, de asignar un valor a una variable y actualizar ese valor cuando sea necesario, están encomendadas en el PASCAL a la llamada sentencia de asignación, cuya forma general es la siguiente:

<variable>: =<expresión>

El operador de asignación es el signo ":=", constituido por dos caracteres. Este asigna el valor situado a su derecha, a la variable que le precede. La zona <expresión> puede ser una constante, una variable previamente declarada y con un valor ya asignado, o bien una expresión matemática o lógica; en este último caso, se trataría de una agrupación de constantes y variables unidas por operadores.
La variable será el nombre de un identificador válido PASCAL; por supuesto, declarado como variable en la zona de definiciones y declaraciones del programa.
La sentencia de asignación lleva implícitas dos funciones. La primera es la de obtener un valor al evaluar la expresión situada detrás del operador ":=", y la segunda la de asignar ese valor a la variable que precede al operador de asignación.
Así, por ejemplo:

CONTADOR := CONTADOR + 1
ESTAVARIABLE := OTRAVARIABLE

TABLA DE COMANDOS-PASCAL		
COMANDO	COMETIDO	OBSERVACIONES
INTEGER	Declaración del tipo entero	Zona de declaraciones y definiciones
REAL	Declaración del tipo real	Zona de declaraciones y definiciones.
CHAR	Declaración del tipo carácter	Zona de declaraciones y definiciones
BOOLEAN	Declaración del tipo booleano	Zona de declaraciones y definiciones

En el primer caso, el valor de la variable CONTADOR es incrementado en una unidad y el nuevo valor se asigna a la propia variable CONTADOR, "borrando" así su valor precedente.
Otro cometido importante de la sentencia de asignación es el de comprobar el tipo de variable. Como ya se indicó, cada variable sólo puede ser de un único tipo y por lo tanto, sólo será aceptada la asignación de una expresión que produzca un valor del mismo tipo que el declarado para la variable.
Hay excepciones a esta regla, puesto que algunos tipos de datos "engloban" a otros. De ahí que, por ejemplo, sea posible asignar un valor entero a una variable declarada como real. En tal caso, será válida dicha sentencia de asignación.
El dialecto de PASCAL más estandarizado no admite la asignación múltiple, o lo que es lo mismo: asignar un mismo valor a una serie de variables. Por ejemplo:

<variable>, <variable2>, ... :=<expresión>

No obstante, algunos compiladores específicos sí permiten tal posibilidad.

EXPRESIONES ARITMETICAS

El objetivo fundamental de una expresión aritmética es el cálculo de un valor que se utilizará como dato. Este será procesado por el programa cuando sea oportuno. Dicho cálculo se efectuará de acuerdo a una serie de reglas que se exponen a continuación. Las expresiones aritméticas se utilizarán en distintas zonas del programa; por ejemplo; en las sentencias de asignación vistas anteriormente, en la definición de constantes, en el argumento de una función estándar, dentro de la condición impuesta en una sentencia de bifurcación condicional...
Una expresión aritmética consiste en un conjunto de operandos, que pueden ser constantes, variables y/o funciones estándar (seno, logaritmo, raíz cuadrada...), unidos por una serie de operadores aritméticos o lógicos, tales como: "+", "*", "OR", ... Por ejemplo, las siguientes son expresiones aritméticas válidas:

SUELDO−((SUELDO*5)/100)
ROUND (PI*R*R)+AREA

En el segundo caso, ROUND es una función estándar que obtiene la parte entera redondeada de un número real.

JERARQUIA DE LOS OPERADORES

Una vez conocidos los diferentes operadores utilizables con cada tipo de datos, veamos cómo hay que manejar estos operadores de forma eficiente.
El valor obtenido al evaluar una expresión matemática puede variar según el orden en el que se ejecuten las operaciones indicadas en la misma. Por ejemplo, en la expresión:

3 * 6 + 2

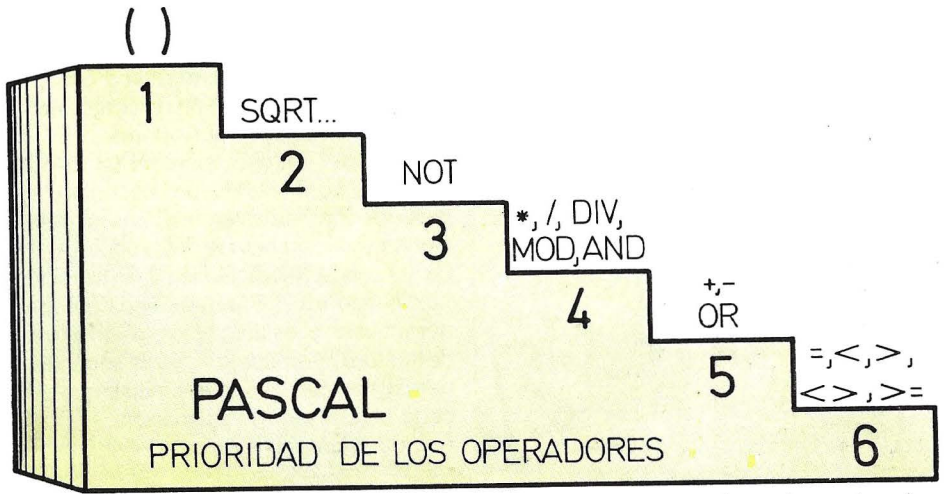
si primero se ejecuta la operación de multiplicación y después la de suma, el valor que se obtiene es 20. Mientras que si primero se efectúa la operación de suma y después la de multiplicación, el resultado será 24.
Para obviar este problema, los lenguajes de los ordenadores establecen un orden jerárquico de operadores, de tal forma que si no se dispone lo contrario, se efectuarán unas operaciones antes que otras.
En el PASCAL, la prioridad y orden jerárquico de los operadores coincide con el siguiente:

1: Funciones estándar
2: NOT
3: *,/,DIV,MOD,AND
4: +,−,OR
5: =,<>,<,>,<=,>=

Cuando en una expresión aparecen varios operadores con un mismo grado de prioridad, se ejecutará el primero de los que se encuentren situados más a la izquierda.
Esta jerarquía puede ser modificada mediante el uso de paréntesis. Las expresiones encerradas entre paréntesis se evaluarán prioritariamente a todas las demás.
En el caso de que existan varios paréntesis "anidados", unos dentro de otros, se evaluarán primero los más interiores. Por ejemplo, la expresión siguiente:

((8*2/4)−1)+(5−ORD('A'))

da como resultado el valor 7.

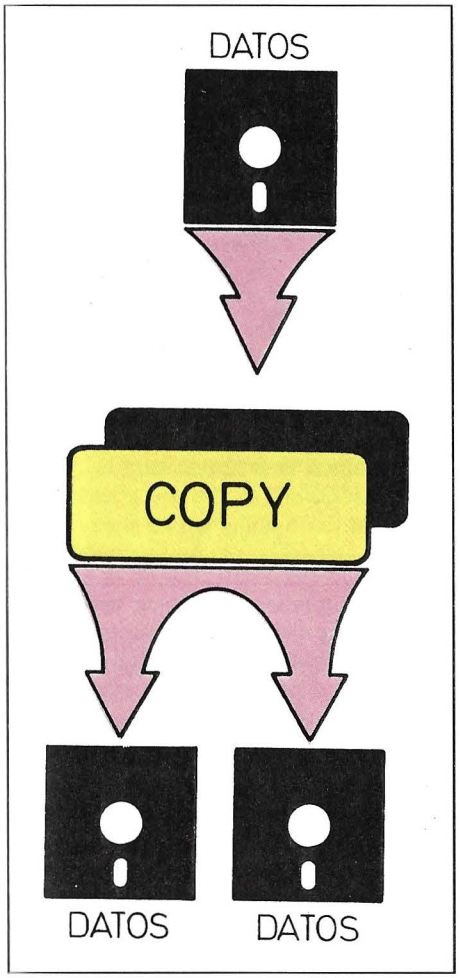


El PASCAL establece un orden de prioridad entre los operadores, este solo puede ser alterado por la presencia de paréntesis.

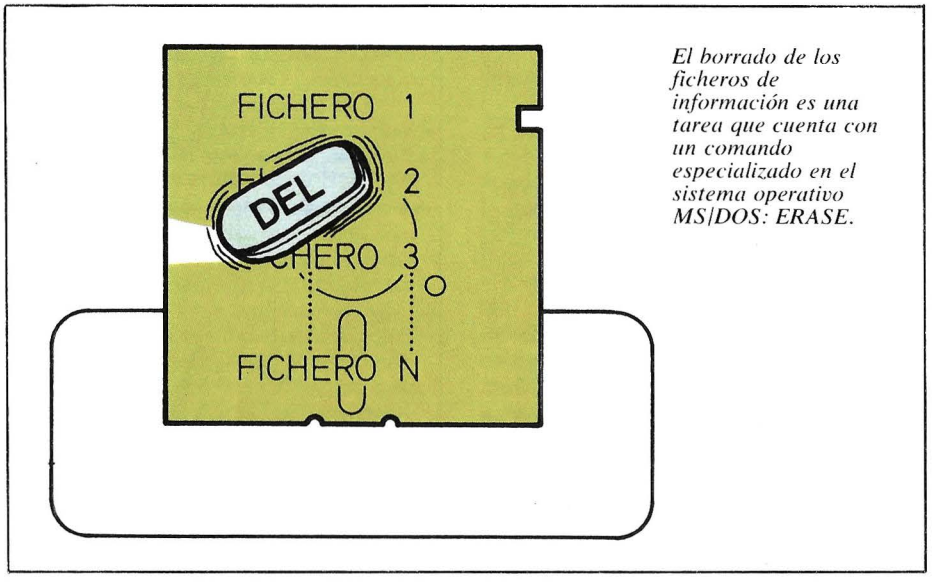
MS/DOS (4)

Comandos avanzados del DOS 1.0

Los comandos incluidos en el presente capítulo se especializan en la operación con ficheros, entendiendo a los ficheros como unidades completas por medio de las que se manipula la información.



La misión del comando COPY se concreta en el duplicado de la información tratada por el sistema. Sus opciones permiten el duplicado de archivos en el mismo disco de origen o en distinto disco.



El borrado de los ficheros de información es una tarea que cuenta con un comando especializado en el sistema operativo MS/DOS: ERASE.

COMANDOS PARA OPERACION CON FICHEROS

Estos comandos actúan sobre los ficheros modificando sus atributos externos, aunque sin modificar en su proceso de operación el contenido interno e intrínseco del fichero. Los comandos adscritos a este grupo son: COPY, DEL, REN y TYPE.

COPY

El comando COPY, tal como su propio nombre indica, se emplea para copiar ficheros. Sin embargo, introduce en esta función tan simple las suficientes variaciones como para servir a tres propósitos distintos:

—Duplicar información en distintos periféricos de almacenamiento masivo.
—Duplicar información bajo otro nombre.
—Combinar información procedente de varios ficheros en un único fichero.

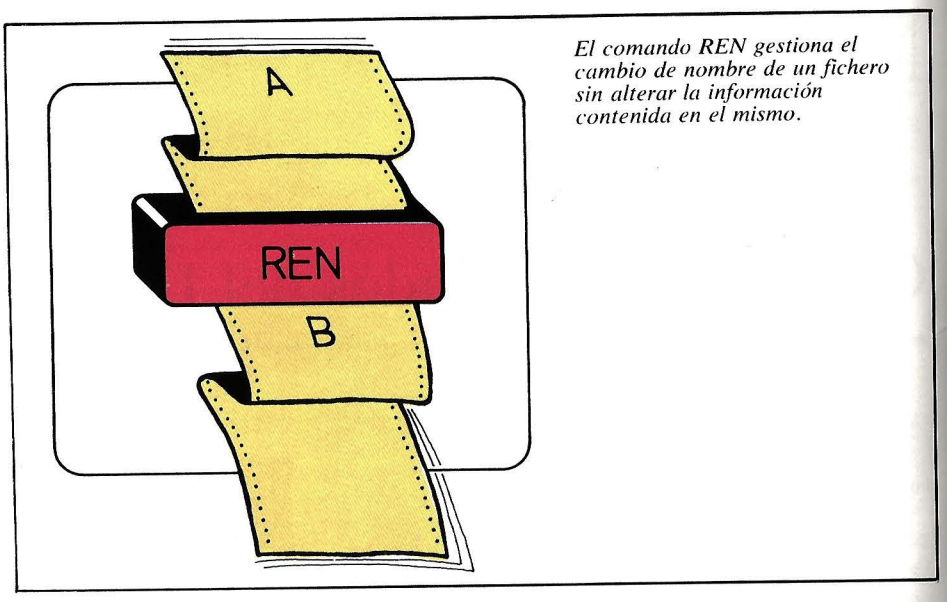
Por lo que se refiere al primero de estos propósitos, el duplicado de información en distintos periféricos, bajo un mismo nombre, el comando COPY necesita que se le defina la información a copiar así como la unidad de disquete sobre la que se va a efectuar dicha copia. Cabe señalar que en este caso, la unidad sobre la que se copia ha de ser distinta de la que contiene el disquete origen de la información. Ello se debe a que, en la versión DOS 1.0, no pueden existir dos ficheros con el mismo nombre sobre un mismo disco.
La forma en que se define el nombre del fichero que se va a copiar es variable, puesto que se admite el empleo de referencias ambiguas o "wildcards". Los ejemplos que siguen ilustran diversas posibilidades.

COPY A:DATOS B:
Este comando producirá una copia del fichero DATOS, contenido en el disquete de la unidad A, sobre el disquete de la unidad B, con el mismo nombre.

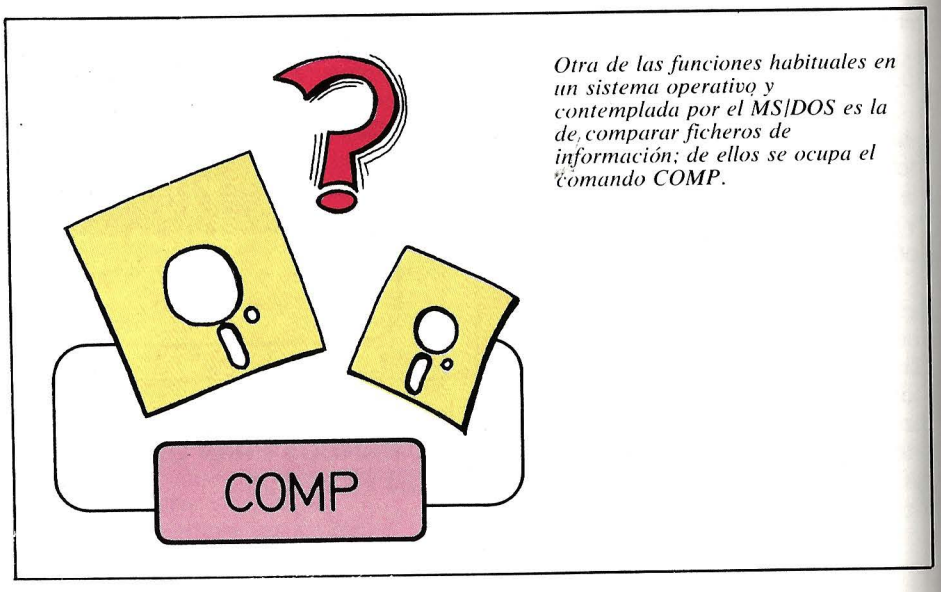
COPY A:DATOS.*B:
En este caso, el empleo de las referencias ambiguas hará que sean copiados los ficheros que tengan como nombre propio "DATOS", sea cual fuere su tipo, y de acuerdo a las reglas indicadas en el caso anterior.
En lo relativo al segundo propósito del comando COPY, el duplicado de información con cambio de nombre, éste se consigue especificando el nombre del fichero que va a ser copiado, así como el nuevo nombre bajo el cual la información va a ser almacenada. De no especificar las unidades de disquete, el sistema operativo asumirá, por defecto, la unidad que en ese momento se considere como la propia del sistema. Por ejemplo:

COPY DATOS RESPALDO
Esta orden producirá una copia del fichero DATOS bajo el nuevo nombre de RESPALDO, residiendo ambos ficheros en el mismo disquete.
El tercer propósito del comando COPY, la agrupación del contenido de varios ficheros en un único fichero, depende en su formato de la versión del MS/DOS empleada. El formato genérico de esta orden suele adoptar el siguiente aspecto:

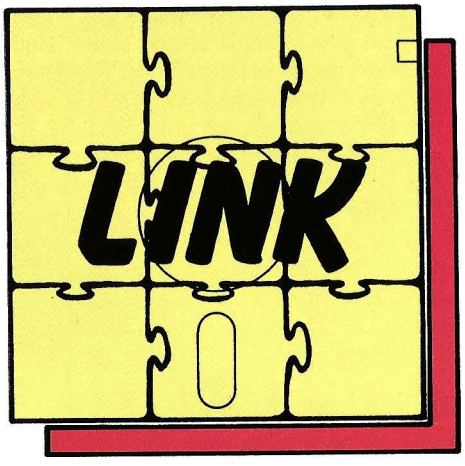
COPY A+B C
Tal y como está expresado, la ejecución del comando hará que los archivos A y B se copien, uno tras otro, en un único fichero resultante denominado C.
COPY es un comando potente y de gran utilidad que, sin embargo, presenta un peligro potencial. Este reside en el hecho de que si existe un fichero con el nombre otorgado a la copia, ésta se producirá sobre el mismo, sin proporcionar al usuario ningún aviso previo. Ello supone que datos valiosos pueden llegar a perderse. Por lo tanto, resulta conveniente emplear previamente el comando DIR, para garantizar que no se ha empleado con anterioridad el nombre del fichero que va a recibir la copia.



El comando REN gestiona el cambio de nombre de un fichero sin alterar la información contenida en el mismo.



Otra de las funciones habituales en un sistema operativo y contemplada por el MS/DOS es la de comparar ficheros de información; de ellos se ocupa el comando COMP.



LINK permite agrupar en un único bloque a los distintos elementos de que consta un programa.

DEL
Este comando se presenta en ciertos equipos bajo el nombre de ERASE, e incluso en otros equipos se admite con ambas denominaciones. Su función es la de borrar o eliminar información. El procedimiento de operación exige definir el fichero que se desea eliminar. Para esta definición pueden emplearse también referencias ambiguas; no obstante, hay que señalar que dependiendo del grado de ambigüedad utilizado, así será el control que tendrá el usuario sobre la operación de borrado. En efecto, una orden del tipo:

DEL DATOS o DEL*.TEST
producirá la eliminación automática y sin ningún tipo de solicitud de confirmación, en un caso del fichero DATOS y en el otro de todos los ficheros cuya extensión del nombre sea TEST. Sin embargo, un comando de la forma:

DEL*.*
solicitará la confirmación para cada uno de los ficheros a eliminar.

REN
El comando REN mantiene una estrecha relación con el comando COPY. Su función es la de cambiar el nombre de un fichero, siempre y cuando el nuevo nombre no sea ya utilizado por otro fichero. Su formato general es el siguiente:

REN NOMBREANTIGUO
NOMBRENUEVO

Al igual que otros comandos, REN también admite el empleo de referencias ambiguas, actuando en tal caso sobre varios ficheros.

REN *.TEST *.PRUEBA

TYPE
La función de TYPE es comprobar el tipo de datos que contiene un fichero, proporcionando una visión del mismo e indicando la naturaleza de los caracteres contenidos (por ejemplo, ASCII).

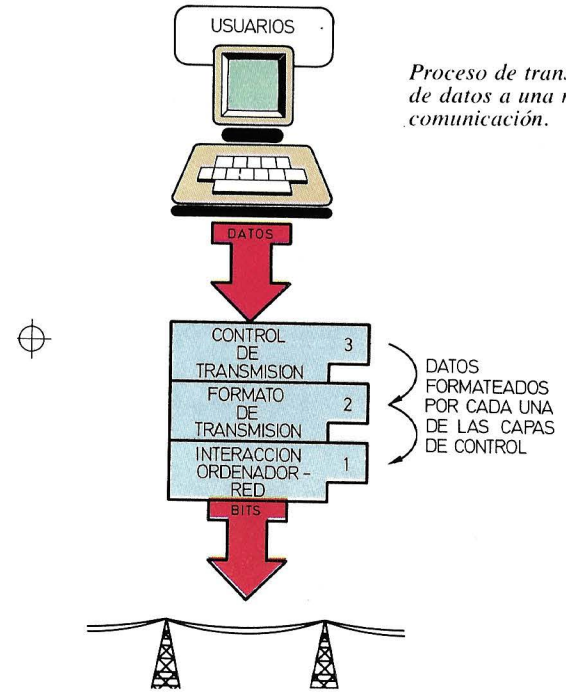
COMANDOS DE COMPARACION

El MS/DOS ofrece dos comandos adecuados para efectuar comparaciones, bien entre ficheros o bien entre disquetes completos. Los comandos al efecto son COMP, el cual compara el contenido de dos ficheros entre sí, y DISKCOMP cuya función es la de comparar el contenido de dos disquetes.

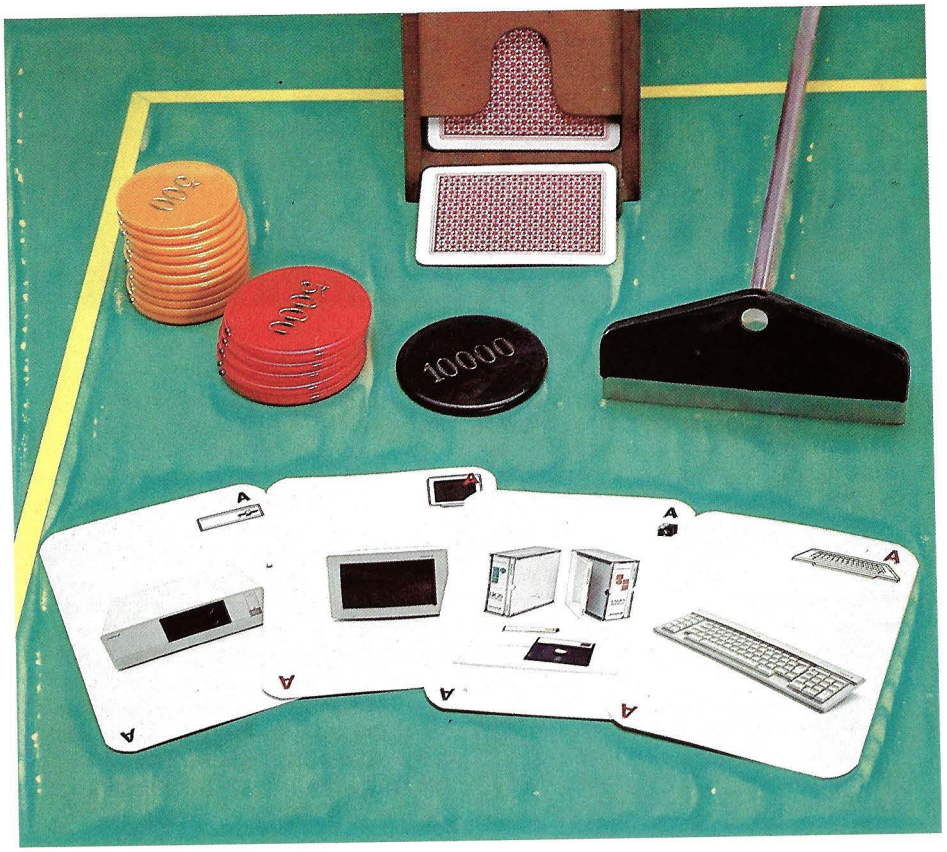
COMP
Este comando es uno de los que presenta mayor variedad de denominaciones en el sistema operativo MS/DOS. Las más comunes, aparte de la ya señalada, son las de FC y FILE COMP.
La función que lleva a cabo es la de comparar entre sí el contenido de dos ficheros, comprobando si ambos coinciden con total exactitud. Esta comparación se efectúa sobre el contenido lógico del fichero, verificando su correspondencia, si bien no tiene en cuenta las posiciones respectivas de los ficheros en el disquete o disquetes en los que residen.
COMP puede presentarse en dos versiones, una "básica" y otra que cabe denominar "avanzada". En la versión "básica", el comando COMP únicamente indica si los dos ficheros sujetos a comparación difieren o no, pero no da ningún tipo de información adicional, con lo que el usuario no sabe cuál es el motivo de las diferencias. En la versión "avanzada" el comando COMP permite señalar una serie de características y hacer caso omiso de otras. En particular, cuando se están comparando ficheros que contienen texto,

La comunicación: un eslabón crítico de la sociedad informatizada

La sociedad informatizada es una idea altamente atractiva y preconizada por muchos estudiosos. En ella la persona trabaja en muchos casos desde su propio hogar, por medio de un ordenador personal conectado a una red que lo comunica con su puesto de trabajo. A través de este canal, el usuario accede a la información del ordenador de su empresa o incluso a la almacenada en otros ordenadores. Ante el auge actual de la informática ésta es una realidad que no parece muy lejana. Sin embargo, hay un problema de base que sólo recientemente se está abordando: la comunicación entre ordenadores distintos, y distantes en su emplazamiento físico.
El proceso de comunicación entre los ordenadores cabe dividirlo en tres capas concéntricas, la primera de las cuales concierne a la relación física entre la máquina y el dispositivo de comunicación. Actualmente, la norma EIA RS/232-C o la recomendación CCIT V.24 brindan un estándar aceptable para este nivel de comunicación.
La segunda capa en el proceso de comunicación está relacionada con la forma de envío de los datos a través de la línea física y con el modo de intercambio de los mismos. En este caso existe también un estándar, el procedimiento HDLC, establecido por la Asociación Internacional de Normas. No obstante, hay que señalar en este punto, que cada fabricante tiene sus propias variantes que difieren ligeramente entre sí.
La tercera capa de la comunicación es el control de la transmisión. Aquí es donde se produce la ruptura, ya que cada fabricante emplea procedimientos de control que le son propios y normalmente incompatibles con los de otros fabricantes; tal es el caso de las arquitecturas SNA de IBM o DECNET de DIGITAL. Los dos niveles inferiores aparecen estandarizados en la mayoría de los casos, aunque bien es cierto que si la forma en que se envían los datos desde el nivel superior es particular del fabricante, poco se ha avanzado. Este es, en definitiva, uno de los puntos críticos en el proceso de construcción de la futura sociedad plenamente informatizada.



Proceso de transmisión de datos a una red de comunicación.



Cumpliendo el cometido propio de un sistema operativo, el MS/DOS pone al alcance del usuario la posibilidad de explotar con eficacia los recursos del ordenador.

este comando permite ignorar si se ha escrito en mayúsculas o minúsculas para la comparación, o bien si existen espacios en blanco. Asimismo permite tomar como origen el punto en el que ambos ficheros difieren y, a continuación, iniciar un proceso de búsqueda hasta encontrar un punto en el que de nuevo vuelvan a coincidir de manera global.
Estas funciones es posible seleccionarlas por medio de interrupciones en el comando, lo cual es, precisamente, la forma de descubrir si la versión del MS/DOS utilizada es la "básica" que no dispone de ellos, o la "avanzada".

DISKCOMP

La misión de DISKCOMP es comparar si dos disquetes son exactamente iguales. El comando no opera en base a una comparación de tipo funcional, sino que también compara físicamente. Es por ello que aunque dos discos contengan exactamente la misma información, pero distribuida de forma distinta en cada uno de ellos, con lo cual serían funcionalmente compatibles y sustituibles, el comando DISKCOMP los señalará como distintos. Este comando sólo tiene interés si existe una necesidad imperiosa de comprobar si dos discos son totalmente iguales.

LOCATE y EXEFIX son dos comandos del MS/DOS cuya especialidad se concreta en la modificación del formato con el que se almacenan los programas.

COMANDOS DE OPERACION CON PROGRAMAS

Los comandos que el MS/DOS destina al manejo de programas, a nivel de sistema operativo, pueden dividirse en dos grupos. Comandos destinados a la reunión de los diversos módulos de un programa y comandos destinados a la modificación de los formatos con que los programas se almacenan en un disco o disquete.

- *Comandos de construcción de programas*

Básicamente son dos: LINK y LIB. El primero de ellos tiene la misión de agrupar el conjunto de elementos que constituyen un programa, como pueden ser el programa principal, las subrutinas que éste utiliza, así como las utilidades necesarias para la operación del mismo; entendiendo por utilidades al conjunto de programas y subrutinas estándar que llevan a cabo determinados trabajos de forma rutinaria. Esta tarea de unión la lleva a cabo el comando LINK, asociado a la operación con lenguajes no interpretados sino compilados.
El comando LIB es un comando menor cuyo cometido es el de mantener en condiciones adecuadas las diferentes librerías de utilidades del usuario, permitiendo la creación de librerías o la eliminación de partes de las mismas.

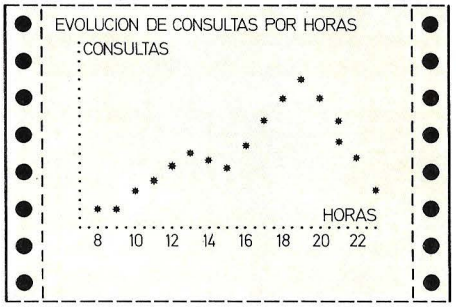
- *Comandos de conversión de formatos*

Cuando un programa se almacena de forma que quede dispuesto para un uso inmediato, lo hace bajo dos posibles formatos; estos son EXE y COM. El formato EXE es el empleado para aquellos programas que en su operación necesitan el pleno empleo de los recursos asociados al MS/DOS, mientras que el formato COM es el empleado con los programas que no exigen este tipo de ayuda; en consecuencia, este último hace un uso más rentable del espacio de almacenamiento en disco.
Dos comandos, LOCATE y EXEFIX, llevan a cabo la tarea de pasar de un formato a otro. LOCATE efectúa el paso del formato EXE a COM, mientras que EXEFIX opera el cambio contrario, de formato COM a EXE.

Software gráfico

Paquetes gráficos para aplicaciones técnicas y de gestión

Dice un viejo refrán que "una imagen vale más que mil palabras" y, en efecto, esto es así en muchos casos. Generalmente los programas de gestión manejan grandes volúmenes de datos apoyándose en ficheros tradicionales o en bases de datos. A partir de esta información, se elaboran informes exhaustivos de los que resulta difícil extraer el "mensaje" que encierran. Un primer paso para facilitar su interpretación consiste en obtener cuadros alfanuméricos resumidos, donde se apreciarán de forma directa los resultados importantes. Sin embargo, la mejor forma de visualizar los datos para su interpretación instantánea es a través de diagramas gráficos. En este capítulo tiene lugar una primera aproximación al software de aplicación estandarizado para crear gráficos de gestión



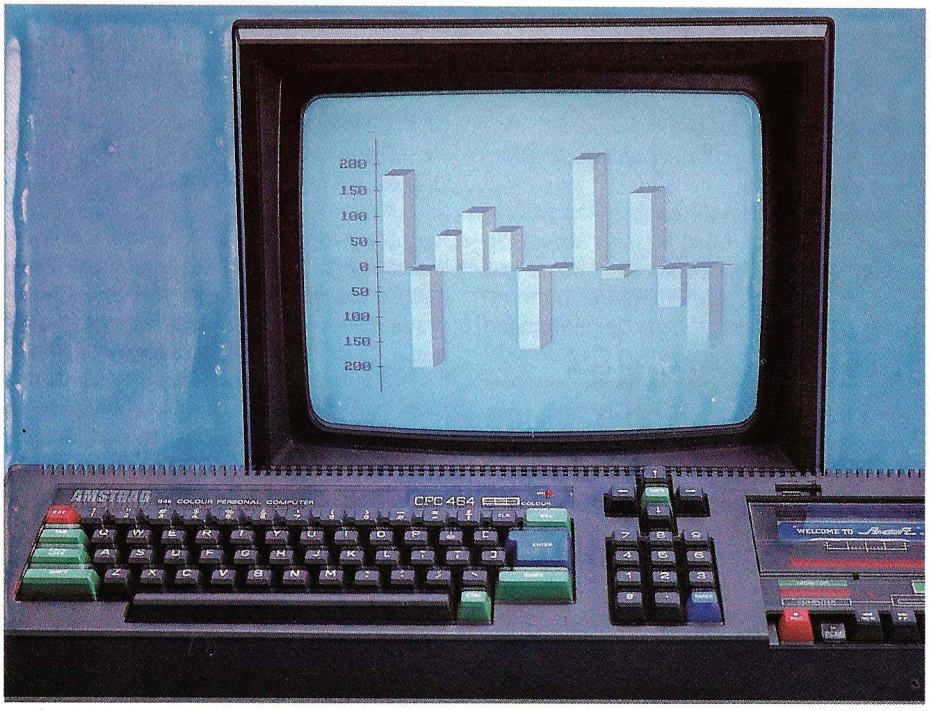
La confección de gráficos por ordenador tiene su expresión más elemental e imprecisa en la obtención del gráfico por impresora, a base de utilizar manualmente algunos caracteres incluidos en el repertorio del equipo.

ANTECEDENTES DEL SOFTWARE GRAFICO

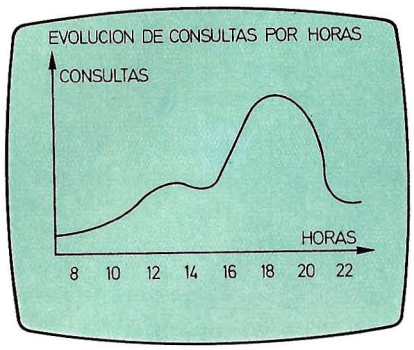
En los primeros años de la informática, prácticamente sólo existía la impresora como periférico de salida para la comunicación ordenador/usuario. Este tipo de dispositivos no eran los más adecuados para la producción de informes gráficos, si bien ya tenían lugar algunos intentos mediante la utilización de caracteres especiales, generalmente el asterisco ("*"); al situarlos convenientemente en el papel continuo, permitían crear representaciones gráficas, aunque con escasa precisión y muy poco atractivo.

El siguiente paso, dentro de lo que cabría denominar informática gráfica, fue la aparición de una nueva categoría de periféricos de salida: los denominados plotters (trazadores gráficos). Estos permitían escribir en papel continuo con plumas o bolígrafos. Evidentemente, ello supuso una revolución en los métodos para la obtención de dibujos a través de un ordenador. No obstante, los plotters sólo cubrían un aspecto parcial de las necesidades gráficas: la vertiente técnica. Los trazadores gráficos están normalmente off-line y, por consiguiente, no permitían la obtención de gráficos de forma inmediata. Además, la entrada de información y su manejo eran lo suficientemente complicados como para asustar a usuarios no informáticos. En resumen, estos equipos se utilizaban fundamentalmente para producir planos técnicos.

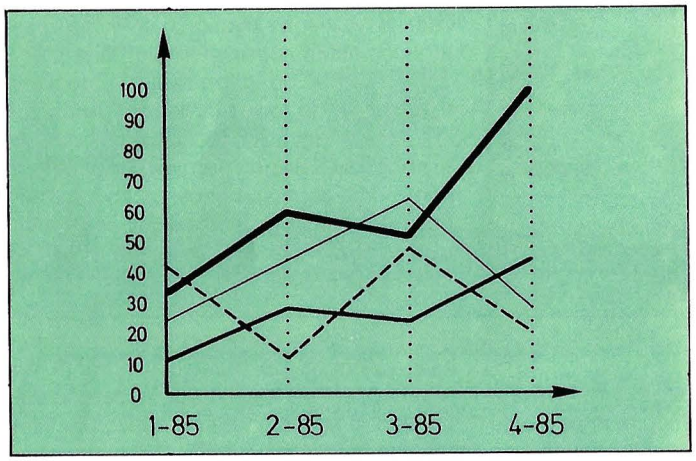
Por fin, hace pocos años, el ámbito de los gráficos de gestión ha encontrado una solución sencilla de manejo y potente en resultados. Para ello ha sido necesario por un lado la aparición de periféricos de alta resolución, tanto pantallas como impreso-



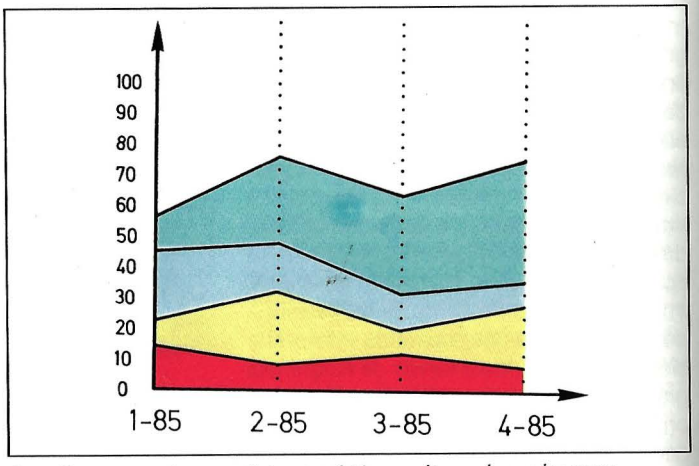
La obtención de gráficos orientados al ámbito de gestión es una de las exigencias más frecuentes en el uso profesional del ordenador.



En la actualidad es posible obtener gráficos en impresoras de alta calidad e incluso sobre pantallas de alta resolución, con una calidad muy aceptable.



Los diagramas de líneas se utilizan para la representación visual de los valores asociados a distintos fenómenos en diversos instantes.



Los diagramas de superficies también resultan adecuados para reflejar el estado de determinados fenómenos en distintas situaciones. En este caso, se superponen, a modo de superficies, los valores medidos para cada fenómeno.

factores; entre ellos, los más importantes son:

1. El periférico fundamental en este tipo de aplicaciones es el plotter, y cada fabricante produce modelos con su propio "sublenguaje" de programación (rutinas de trazador).
2. Los principales usuarios son empresas de ingeniería que tradicionalmente han desarrollado su propio software.
3. Las necesidades están muy diversificadas, de forma que cada programa desarrollado tendría pocos compradores y, por lo tanto, sería de difícil amortización.

En la actualidad se está produciendo un cambio de enfoque en este sector, sustituyendo a la pareja programa/plotter por la pareja delineante/sistema gráfico.

Uno de los principales problemas que han dificultado el desarrollo de buenos programas para la producción de planos ha sido la enorme casuística presentada. La variedad de los problemas era tan grande que resultaba prácticamente imposible recoger a todos ellos en un programa único. Por lo tanto, la solución que se está imponiendo es completamente distinta a la sustitución de un delineante por un programa (en el fondo, tal era el objetivo que se perseguía), y se basa en la creación de sistemas especializados que sirvan como herramienta al técnico, permitiendo que realice su trabajo con mucha más precisión y rapidez.

En definitiva, un sistema especializado en tareas de diseño consiste en una pantalla, basada en un tubo de rayos catódicos de alta resolución, y una superficie situada delante de ella (sustitutiva del teclado) en la que se puede "pintar" con un lápiz especial. Según el punto de la superficie sobre el que actúe el lápiz, aparecerá en la pantalla una determinada línea o presentación gráfica. De esta forma, el operador puede ir creando los dibujos sobre la pantalla. Cuando haya terminado el plano, puede almacenarlo en algún dispositivo de memoria auxiliar, o imprimirlo mediante un plotter.

En algunos casos existe la posibilidad adicional de utilizar un periférico denominado *digitalizador,* cuyo cometido es la representación mediante dígitos de un dibujo. Con su colaboración, los puntos singulares del gráfico tratado se convierten en coordenadas que pueden ser manejadas eficientemente mediante programas tradicionales.

Por último, dentro del software gráfico para aplicaciones técnicas, podemos citar una metodología de trabajo que aún está en investigación y cuyo empleo se limita a algunos casos. Dicha metodología consiste en realizar una digitalización no de los puntos singulares del gráfico, sino de todos y cada uno de sus puntos elementales (en la pantalla se denominan pixels). El valor asignado a cada uno de ellos se reproducirá con un tono de gris si se trabaja en blanco y negro (pantallas monocromáticas), o con un determinado tono y color si se trabaja con colores (pantallas policromáticas). Mediante esta técnica se han logrado éxitos contundentes en la restitución de imágenes a partir de una fotografía defectuosa, bien porque su enfoque fuera deficiente o bien porque apareciera movida. La fotografía en cuestión es digi-

ras, y por otro la existencia de ordenadores personales que han acercado la informática al usuario.

En resumen, cabe afirmar que los paquetes de software gráfico tiene su ámbito de acción en dos campos específicos: los gráficos técnicos (generalmente denominados planos) y los gráficos de gestión (en adelante los denominaremos diagramas).

SOFTWARE GRAFICO TECNICO

Los programas de esta categoría empezaron a llegar al mercado hace ya algunas décadas; si bien, un hecho constatable es que no existen demasiados paquetes estandarizados. El motivo obedece a varios

talizada; a partir de estos datos y mediante un tratamiento matemático de los dígitos, se puede obtener una reproducción de calidad muy superior a la del original.

SOFTWARE GRAFICO DE GESTION

Los programas gráficos destinados a satisfacer necesidades de gestión, no comparten la problemática enunciada para el caso de los programas técnicos. Ahora, el objetivo se reduce a representar en un sencillo diagrama unos datos numéricos, de tal forma que sin más que observar el gráfico queden patentes las propiedades de los datos. La mayoría de los paquetes ofrecen varias alternativas de representación; las cuatro más comunes son los diagramas de *líneas,* los diagramas de *superficies,* los diagramas de *tarta* y los diagramas de *barras.*

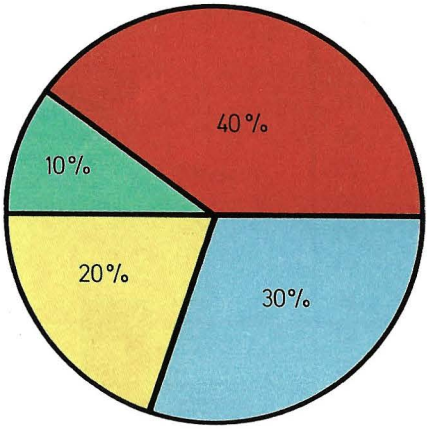
- *Diagramas de líneas*

Probablemente sea éste el más extendido y popular de los métodos de representación gráfica. Se fundamenta en dos ejes denominados de abscisas y de ordenadas. El primero, dispuesto en una línea horizontal, identifica los "momentos" que se representan en la vertical de cada punto. En este eje suele representarse, precisamente, el tiempo en cualquiera de sus medidas: horas, días, semanas, meses, trimestres, años... El segundo eje, trazado en sentido vertical, indica los valores concretos del fenómeno estudiado. Aquí la unidad de medida no es tan homogénea como el caso de las abscisas; en el eje de ordenadas se pueden medir: pesetas, porcentajes, Kg... y, en general, prácticamente cualquier unidad de medida.
En el espacio comprendido entre los dos ejes se trazan una serie de líneas, de tal forma que cada punto de una línea corresponde a un momento concreto (identificado por su proyección vertical en el eje de abscisas) y al correspondiente valor del fenómeno en ese instante, valor dado por la proyección horizontal de dicho punto sobre el eje de ordenadas.
Los diagramas de líneas se pueden dividir en dos grupos:

1. De una sola línea.

Se utiliza para representar de forma aislada un único fenómeno.

El diagrama de tipo tarta es tal vez el más visual. Resulta muy adecuado para representar el estado de distintos fenómenos en un instante único.



ENERO-85

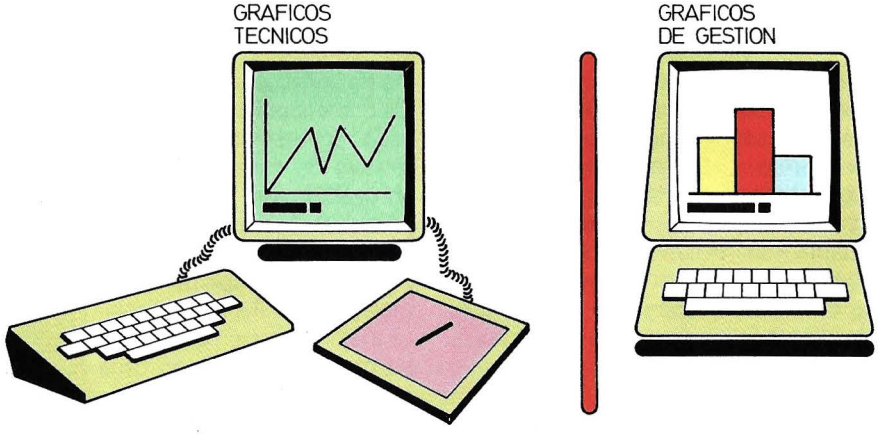
2. De varias líneas.

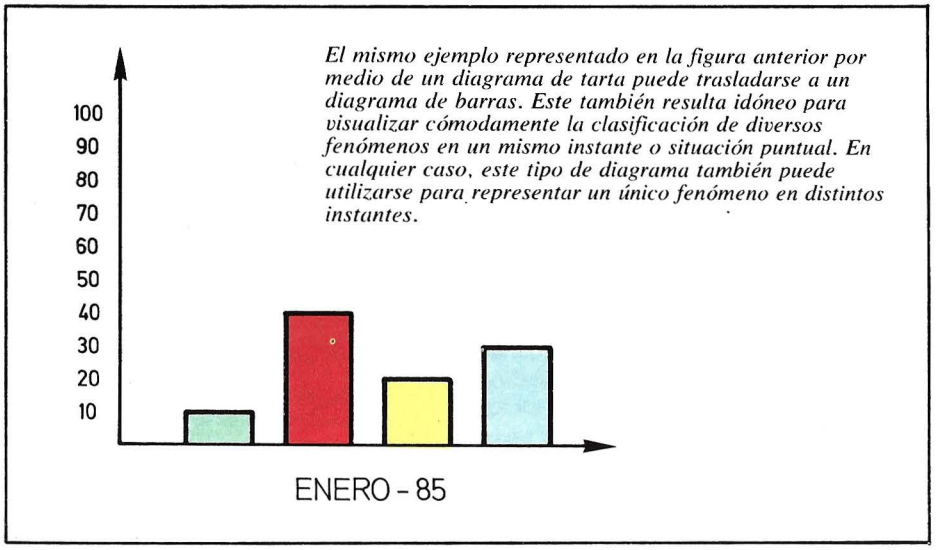
Al superponer en el mismo diagrama varias líneas, cada una de las cuales representa a un fenómeno específico, no sólo se observa cada uno de los fenómenos sino que también se apreciará la relación existente entre ellos.
El principal problema que origina la representación de varias líneas en un único diagrama es la dificultad de distinguir cada una de las líneas, sobre todo cuando se cruzan varias de ellas. Para evitar esto, se suelen utilizar distintos colores o distintos tipos de línea (continua, discontinua, punteada, etc.).

Terminales gráficos

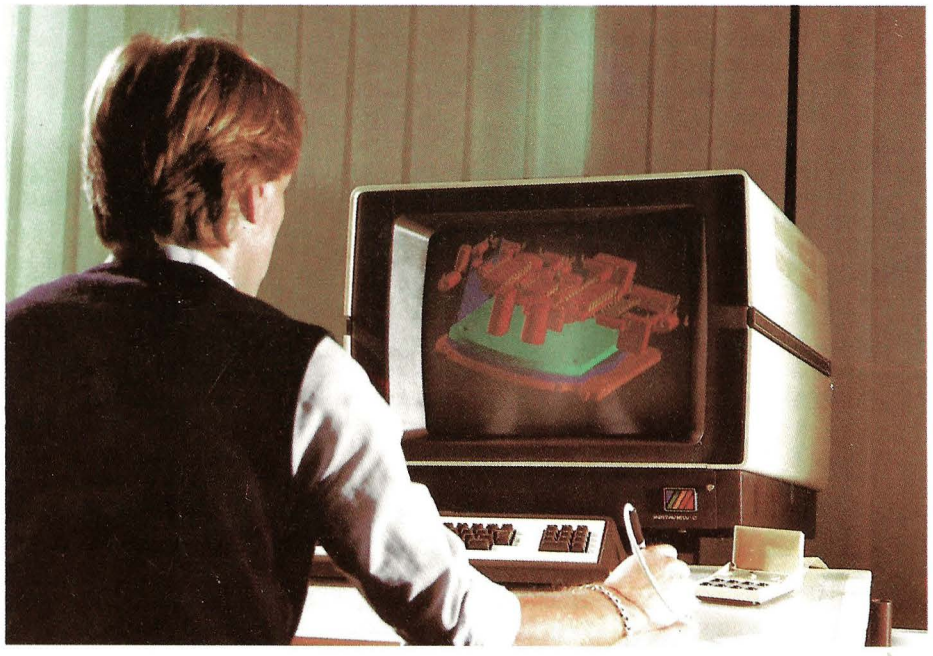
Existen básicamente dos tipos de terminales gráficos: los monocromáticos (un solo color sobre el fondo de la pantalla) y los policromáticos (de color). Los primeros se suelen reservar a aplicaciones técnicas, mientras que los segundos son más frecuentes en aplicaciones de gestión. No obstante, gracias a la caída de precios que está experimentando todo el hardware en general, se tiende a trabajar en ambos casos con pantallas de color.
Las principales características que permiten diferenciar a los terminales gráficos técnicos de los terminales de gestión se reflejan en el siguiente cuadro adjunto.

	TERMINALES GRAFICOS TECNICOS	TERMINALES GRAFICOS DE GESTION
Tamaño	Grandes/Medianos	Medianos/Pequeños
Color de la pantalla	Monocromáticos	Policromáticos
Resolución de la pantalla	Alta/Media	Media/Baja
Teclado u órgano de entrada de datos	Teclado tradicional y tableta gráfica	Teclado tradicional
Usuario	Operador	Operador/Programas
Soporte	Ordenador especializado (no siempre)	Ordenador convencional (Mini o microordenador)





El mismo ejemplo representado en la figura anterior por medio de un diagrama de tarta puede trasladarse a un diagrama de barras. Este también resulta idóneo para visualizar cómodamente la clasificación de diversos fenómenos en un mismo instante o situación puntual. En cualquier caso, este tipo de diagrama también puede utilizarse para representar un único fenómeno en distintos instantes.



Las posibilidades de creación y tratamiento gráfico encuentran su máxima expresión en los sistemas destinados al diseño y control asistido por ordenador (sistemas CAD/CAM).

grama, radica en la posible diferencia entre las unidades de medida. Si una de ellas tiene valores muy superiores a las demás, esta aparecerá aislada en la parte superior; en cambio, las restantes se representarán muy próximas en la zona inferior del diagrama. Para evitar este inconveniente, se pueden utilizar distintas escalas en los ejes de un mismo diagrama, o separar la representación en diagramas distintos.

- *Diagramas de superficie*

Los diagramas de superficie, al igual que los de líneas, se apoyan en dos ejes entre los que estratifican la información. En este caso representan mediante áreas el valor relativo de cada uno de los fenómenos analizados; para ello, en el eje de abscisas se ubican los distintos momentos en los que se dispone de medidas y en el de ordenadas se superponen de forma acumulativa los valores medidos.
La mejor aplicación de este tipo de gráficos se obtiene cuando, en cada período, la suma de los valores medidos es constante, de tal forma que al analizar las distintas superposiciones se obtiene un rectángulo. Usualmente, el eje de ordenadas se mide en porcentajes, con lo que se cumple la propiedad anterior.

- *Diagramas de "tarta"*

En este caso no se utiliza ningún tipo de ejes; precisamente por ello, los diagramas de tartas resultan los más "visuales". La representación adopta la forma de un círculo, en cuyo interior se marcan las distintas porciones que corresponden a los valores medidos de los fenómenos analizados.
Normalmente, las medidas aparecen en porcentajes. De esta forma, la "tarta" completa representa el 100% y las porciones representan la parte proporcional que corresponda en cada caso. Los dos tipos de diagramas descritos anteriormente (de líneas y superficies) venían determinados por pares de valores: un valor indicaba el momento de medida y el otro la magnitud medida. En cambio, en los diagramas de tarta, sólo se tienen en cuenta los valores medidos y, por lo tanto, la información contenida en una tarta hace referencia a un único instante.

- *Diagramas de barras*

Al igual que en el caso anterior, sirven para reflejar gráficamente las medidas de varios fenómenos en un único instante (o un único fenómeno en varios instantes); de ahí que, normalmente, sólo utilicen un eje de valores. A pesar de ello, acostumbran a representarse entre dos ejes coordenados. El de abscisas suele servir exclusivamente de soporte para las barras, una por cada fenómeno representado; éstas tendrán una altura acorde con el valor medido para dicho fenómeno.
La utilidad de los diagramas de barras es muy similar a la de los gráficos de tarta, con la salvedad de que mediante las barras se pueden tomar medidas en cualquier unidad, en cambio, mediante las tartas la unidad casi obligatoria es el porcentaje.
En el caso de que se pretenda representar mediante un único diagrama información de varios fenómenos en distintos momentos, se puede utilizar una variedad de los diagramas de barras denominada *multibarra*. Simplemente consiste en utilizar el eje de abscisas para representar en él los distintos momentos y en cada uno de ellos se sujeterá una barra para cada uno de los fenómenos.

Funciones matemáticas

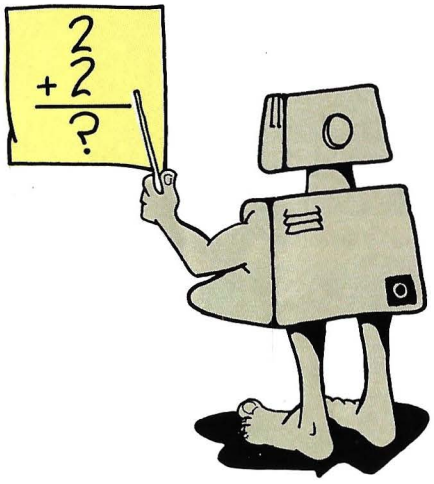
Las herramientas de cálculo del lenguaje BASIC

Los ordenadores nacieron como una herramienta de cálculo. La historia de la informática se remonta a las antiguas máquinas de calcular mecánicas. Quizás el más antiguo predecesor de los modernos ordenadores sea el ábaco, con el que se podían realizar sumas, restas y hasta multiplicaciones.

Los primeros ordenadores electrónicos no iban mucho más allá. Hoy en día la actividad del ordenador abarca prácticamente todos los campos, pero sin duda, el cálculo matemático es uno de los principales.

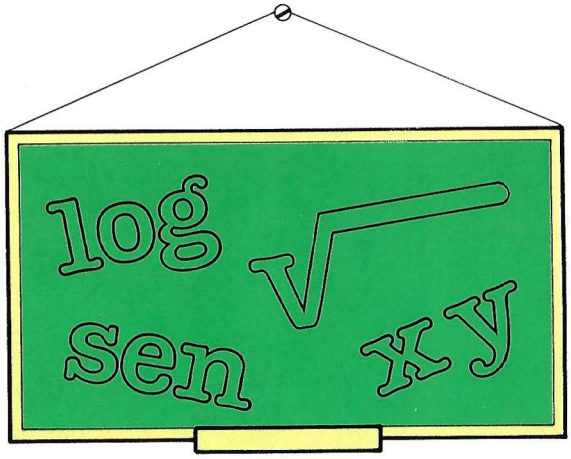
Todos los lenguajes de programación incluyen, entre su repertorio de instrucciones, funciones de tipo matemático. El BASIC es un lenguaje derivado de otro más antiguo: el FORTRAN. Este último estaba claramente orientado a los problemas de índole científica y matemática, y por ello aportaba funciones matemáticas de fácil uso. El BASIC ha heredado del FORTRAN esta facilidad para el tratamiento de datos numéricos.

En capítulos anteriores se ha hablado de los operadores matemáticos elementales.



Con ellos se podían realizar operaciones tales como suma (+), resta (−), multiplicación (*), división (/) y potenciación (^ ó ↑). Se vio también su uso en sentencias de asignación y como parámetros calculados en el argumento de un comando. Los siguientes ejemplos muestran estas posibilidades.

```
LET S=A^2+5*B
PRINT A^2+5*B
```



Las funciones matemáticas del BASIC son las herramientas fundamentales que permiten instruir al ordenador para que resuelva complejos problemas.

La capacidad de cálculo en BASIC no queda ahí. Este lenguaje está capacitado para realizar operaciones más complejas.

RAICES CUADRADAS

El lenguaje BASIC dispone de una función específica para el cálculo de raíces cuadradas. Esta función ha sido utilizada con anterioridad dentro de la obra, para realizar algunos pequeños cálculos.

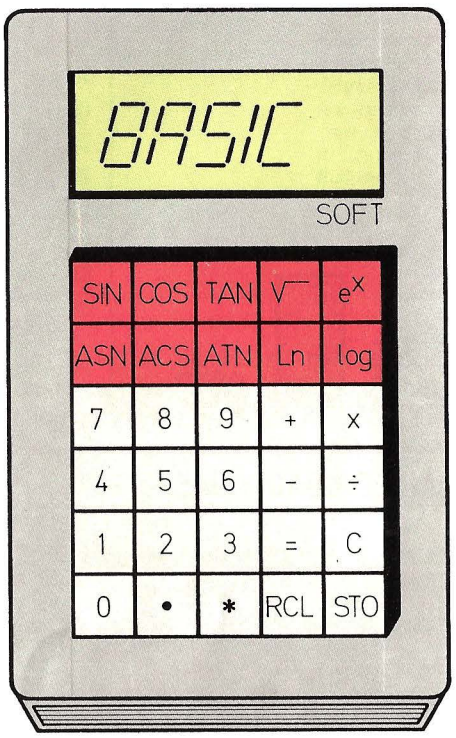
Su formulación es muy sencilla, se apoya en la palabra clave SQR (del inglés SQuare Root, raíz cuadrada). Dicha palabra clave ha de ir seguida por el argumento, que no es más que el dato del que se desea extraer la raíz, situado entre paréntesis. Así pues, para hallar la raíz cuadrada de 2 basta con escribir SQR(2).

Esta, al igual que las restantes funciones de tipo matemático, debe estar acompañada por un comando BASIC, por ejemplo puede ir precedida por el comando LET dentro de una sentencia de asignación:

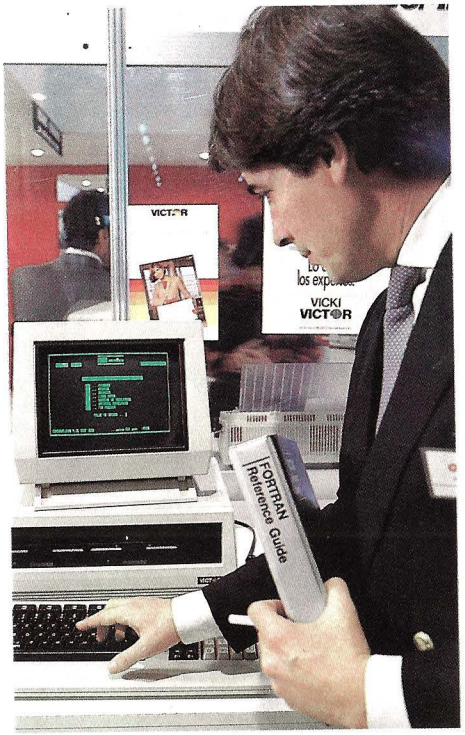
```
10 LET RAIZ=SQR(2)
```

El argumento de SQR ha de contener un dato numérico entre paréntesis. Este dato puede pertenecer a cualquiera de los tipos normales en BASIC: constante, variable o expresión. Así, se puede indicar el cálculo de la raíz cuadrada de una variable, con lo que se consigue una mayor flexibilidad de uso.

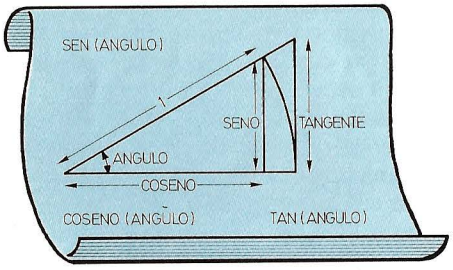
Las expresiones matemáticas introducidas en el argumento pueden incluir, normalmente, cualquier operador o función metamática BASIC (incluso otra raíz cuadrada); por ejemplo:



El repertorio de posibilidades de cálculo que brindan las funciones matemáticas del BASIC no distan mucho de las habituales en una calculadora científica.



El lenguaje BASIC ha heredado del FORTRAN su facilidad para el tratamiento de datos numéricos.



La resolución de cálculos trigonométricos es una de las vertientes contempladas en el lenguaje BASIC. Es este aspecto, las funciones básicas para el cálculo del seno, coseno y tangente (SIN, COS y TAN) están presentes en casi cualquier dialecto BASIC.

```
LET RAIZ=SQR(A)
LET SOLU=SQR(6*A+1/2)
PRINT SQR(2+SQR(A/B))
```

OTRAS FUNCIONES

La utilización de la función SQR es casi imprescindible para la resolución de cualquier problema matemático. Al profundizar algo más en el cálculo matemático se hace necesario el uso de funciones más complejas. El siguiente paso son las funciones trigonométricas. En BASIC se puede hacer uso del seno, coseno y tangente mediante las palabras clave SIN, COS y TAN respectivamente. Al igual que la función SQR, éstas llevarán su dato de entrada entre paréntesis. El dato puede ser de cualquiera de los tres tipos anteriormente enumerados.

```
LET A=SQR(SIN(X)+COS(X))
PRINT TAN(A+2*B)
```

Estos dos ejemplos ilustran las posibilidades de las funciones trigonométricas. El ángulo sobre el que actúan se debe dar, por regla general, en radianes. De esta forma, para calcular el seno (coseno o tangente) de un ángulo expresado en grados, hay que pasar dicho ángulo previamente a radianes. En todo caso, se puede escribir un pequeño programa que realice tal conversión. Lo único que hace falta saber es que 180 grados corresponden a (número PI) radianes. Con este dato y mediante una sencilla regla de tres, se consigue la fórmula que permite pasar de grados a radianes: radianes=grados *(/180).

```
10 REM PASO DE GRADOS A RADIANES
20 LET PI=3.1416
30 INPUT "GRADOS";G
40 PRINT G*(PI/180);" RADIANES"
50 END
■
```

En este programa se incluye la fórmula de conversión directamente en el comando PRINT de la línea 40. Es importante introducir el valor de π en la correspondiente variable, tal y como se hace en la línea 20. Algunos aparatos disponen de la constante π directamente. En tal caso se puede acceder a su valor poniendo sencillamente su símbolo (π) o, en otros equipos, el nombre (PI). Con esta facilidad el programa se reduciría a la expresión siguiente:

```
10 REM PASO DE GRADOS A RADIANES
30 INPUT "GRADOS";G
40 PRINT G*( /180);" RADIANES"
50 END
■
```

SQR

Halla la raíz cuadrada del dato situado en su argumento.

Formato: SQR (<dato>)

Ejemplos: 10 LET A=SQR(17)
50 PRINT SQR(B+C)

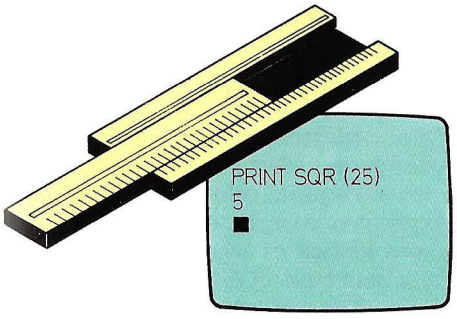
Tanto de una como de otra forma el programa proporciona el valor en radianes apto para ser incluido en el argumento de cualquier función trigonométrica. Su ejecución mostrará por pantalla un aspecto similar al siguiente:

```
RUN
GRADOS?45<CR>
,7853981675 RADIANES
■
```

De esta forma se puede hacer que el ordenador calcule senos, cosenos o tangentes de un ángulo dado en grados.
Esta misma filosofía permite construir funciones que trabajen directamente con grados. El siguiente ejemplo muestra cómo realizar un programa que calcula el seno de un ángulo cuyo valor se introduce en grados. Para el coseno o la tangente la actuación sería similar.

```
10 INPUT "SENO ";G
20 PRINT SIN(G* /180)
```

La expresión encerrada entre paréntesis es la que transforma los grados en radianes. Algunos ordenadores permiten elegir el modo de operación (grados o radianes). Para ello disponen de dos comandos: DEG y RAD. El primero hace que los ángulos se tomen en grados. Así, las funciones operarán en lo sucesivo con su argumento en grados. El último ejemplo se simplificaría adoptando ahora el siguiente aspecto:



Con la ayuda de las funciones de esta categoría, el ordenador será capaz de suplir a cualquier sofisticado utensilio manual de cálculo con eficacia y comodidad para el programador.

SIN

Calcula el valor del seno natural del ángulo incluido en su argumento.

Formato: SIN (<ángulo>)

Ejemplos: 20 LET SENO=SIN(ANG)
70 PRINT SIN(2*X+1)

COS

Calcula el valor del coseno natural del ángulo incluido en su argumento.

Formato: COS(<ángulo>)

Ejemplos: 20 LET CON=COS(ANG)
70 PRINT COS(B+90)

TAN

Calcula el valor de la tangente del ángulo incluido en su argumento.

Formato: TAN(<ángulo>)

Ejemplos: 20 LET TGN=TAN(ANG)
70 PRINT TAN(I−45)

ATN

Calcula el arco cuya tangente es el valor que aparece en su argumento.

Formato: ATN(<dato>)

Ejemplos: 20 LET ARCO=ATN(X)
70 PRINT ATN(A+B/2)

```
 5 DEG
10 INPUT "SENO ";G
20 PRINT SIN(G)
```

El comando RAD restituye el modo de operación a radianes.
El resto de las funciones trigonométricas (secante, cosecante y cotangente) pueden ser calculadas a partir de estas tres. Su equivalencia es la siguiente:

```
Secante (X)=1/COS(X)
Cosecante (X)=1/SIN(X)
Cotangente (X)=1/TAN(X)
```

Cuando se utilizan las funciones trigonométricas elementales se hace necesario disponer de sus inversas (arcoseno, arcocoseno y arcotangente).
Estas son las que realizan la acción contraria a las anteriores. Así, si X=seno (A), sucede que A=arcoseno (X).
En BASIC estas funciones se formulan de la siguiente forma: ASN (arcoseno), ACS (arcococoseno) y ATN (arcotangente). Su utilización es idéntica a la de SIN, COS o TAN.

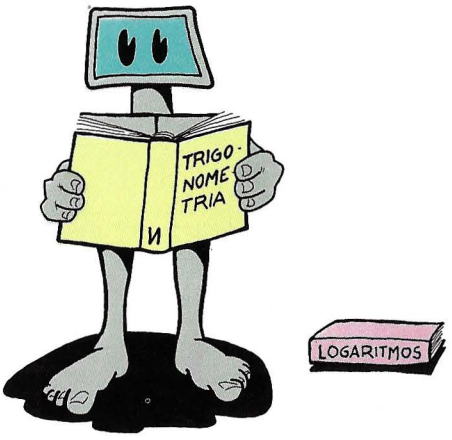
```
50 LET A=ASN(0,5)
10 PRINT ACS(X/SQR(2))
```

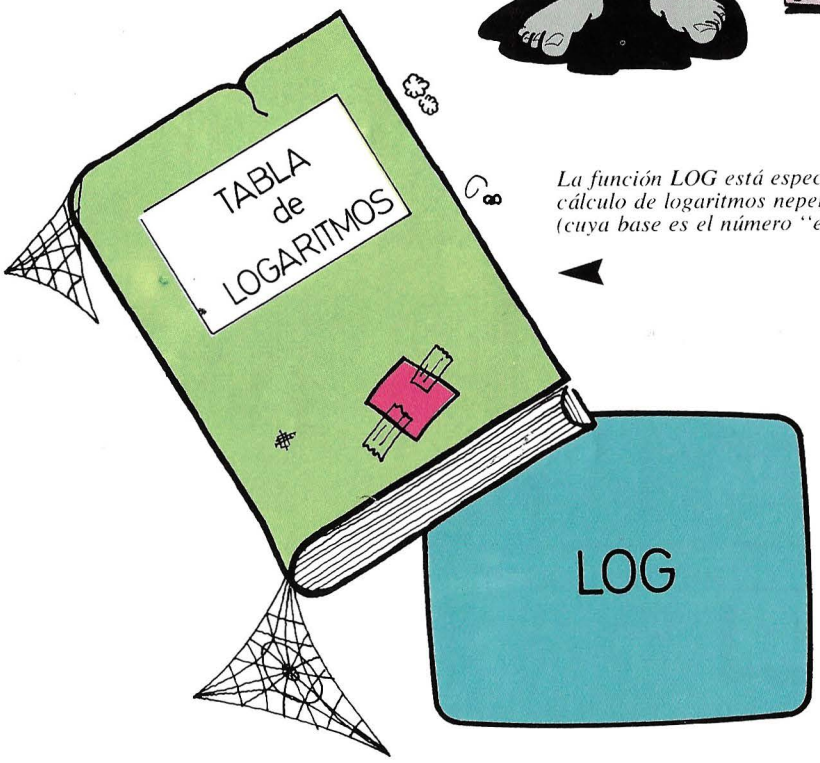
En algunos aparatos tan sólo se dispone de una de las funciones inversas, por lo

general ATN. Ello no plantea ningún problema ya que las otras dos son calculables a partir de ésta. Las fórmulas que relacionan estas funciones entre sí son:

```
ASN (X)=ATN (X/SQR(1-X^2))
ACS (X)=ATN (SQR(1-X^2)/X)
```

La deducción de estas fórmulas es sumamente fácil con unos ligeros conocimientos de trigonometría.





La función LOG está especializada en cálculo de logaritmos neperianos (cuya base es el número "e").

DEG

Permite el cálculo de las funciones trigonométricas expresando los ángulos en grados.

Formato: (número de línea) DEG

Ejemplos: 30 DEG

RAD

Especifica, en el cálculo de funciones trigonométricas, que los datos se dan en radianes.

Formato: (número de línea) RAD

Ejemplos: 20 RAD

LOGARITMOS

Tan importante como el uso de las funciones trigonométricas es el de los logaritmos.
La mayoría de los intérpretes BASIC poseen una función capaz de calcular logaritmos. Esta función suele ser LOG, que da el valor del logaritmo neperiano o natural del dato incluido en su argumento. Este logaritmo es el que tiene como base el conocido número **e** (2,718281...).
Los siguientes son ejemplos válidos de uso de la función LOG:

```
PRINT LOG (T+14)
LET X=5*LOG (17)
```

Partiendo del concepto de logaritmo:

$\text{Log}_b x = y$

en donde se cumple que:

$x = b^y$

se observa que la función inversa resulta muy fácil de obtener. En las expresiones precedentes, b es la base (en este caso el número e). De ahí que para calcular el valor de x, conocido el valor de su logaritmo (y), será suficiente con elevar el número "e" (la base) a la potencia indicada (y). En definitiva, para hallar, en BASIC, el antilogaritmo de un número Y servirían las siguientes líneas.

```
10 LET E=2.71828182
20 LET ANTILY=E^Y
```

No obstante, la mayor parte de los intérpretes BASIC poseen una función adecuada para su cálculo directo. Se trata de la función EXP. Así pues, EXP(Y) equivale a 2.71828182^Y. De esta forma, el anterior ejemplo quedaría reducido a:

```
20 LET ANTILY=EXP(Y)
```

Otros ordenadores disponen también de una función adecuada para hallar logaritmos decimales, o lo que es lo mismo: logaritmos cuya base es el número **10.** Dicha función es accesible por medio de la palabra clave LOG10.
El hecho de que sean pocos los ordenadores que incluyen esta última función no plantea inconveniente alguno. Es perfectamente posible calcular logaritmos de cualquier base partiendo de los neperia-

nos. Para deducir la fórmula que permite expresar logaritmos de una base cualquiera b mediante logaritmos neperianos, basta con conocer una simple propiedad de los logaritmos:

$k \cdot \text{Log } x = \text{Log } x^k$

Así pues, si y es el logaritmo decimal de un número x, resulta cierto que:

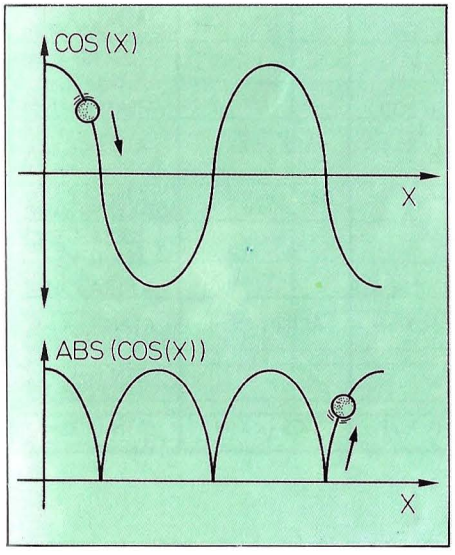
$y = \text{Log}_{10} x \Rightarrow x = 10^y$

Si se aplican logaritmos a los dos miembros de la segunda expresión (Log_e=Ln) queda:

$\text{Ln } x = \text{Ln } 10^y$

Según la propiedad anteriormente mencionada, se puede sustituir el segundo miembro de la igualdad.

$\text{Ln } x = y * \text{Ln } 10$



Representación gráfica de las funciones COS (X) y ABS (COS(X)). Por medio de ellas es posible simular la trayectoria de una pelota botando.

Ahora sólo queda despejar la variable y, que coincide con el valor del logaritmo en base 10 de x. Con ello se obtiene la fórmula buscada:

$$y = \text{Log}_{10} x = \frac{\text{Ln } x}{\text{Ln } 10}$$

Si lo que se desea es calcular logaritmos de otra base, basta pues con indicar la nueva base en el argumento del logaritmo neperiano situado en el denominador de la expresión. Esta fórmula se codificaría en BASIC como sigue:

```
LET LBX=LOG(X)/LOG(B)
```

En la variable B debe encontrarse almacenado el valor de la base deseada. De esta forma, en LBX se obtiene el logaritmo en base B del número X.

LOG

Calcula el valor del logaritmo neperiano del dato dado en su argumento.

Formato: LOG(<dato>)

Ejemplos:
```
10 LET LGX=LOG(X)
75 PRINT LOG(14*S+2)
```

EXP

Calcula el valor del antilogaritmo del número incluido en su argumento. (e ↑ X).

Formato: EXP(<dato>)

Ejemplos:
```
30 LET ANTI=EXP(K)
120 PRINT EXP(-X/K)
```

MATEMATICAS DIVERTIDAS

Las funciones matemáticas expuestas serán de gran utilidad para la resolución de problemas sofisticados. Cualquier estudio avanzado, dentro de cualquier disciplina, desde la economía hasta la física, hace uso de los logaritmos y de las funciones trigonométricas. Sin embargo, no es necesario apelar a la alta matemática para dar un ejemplo del uso de estas funciones. Incluso es posible conseguir efectos divertidos con las funciones estudiadas.
Por ejemplo, se puede simular la trayectoria de una pelota botando por medio de las funciones trigonométricas. Para ello se utilizará la función coseno, aunque bien es cierto que se podría utilizar el seno con parecido resultado. Más concretamente, y a la vista de las respectivas representaciones gráficas, se hará uso del valor absoluto del coseno.

Al variar el valor de X en la expresión ABS (COS(X)) se van obteniendo diferentes datos. Estos datos pueden asimilarse a las distintas alturas que toma la pelota. Por medio del comando PRINT se puede plasmar esa trayectoria en la pantalla. Las coordenadas X e Y del movimiento se corresponderán con los dos parámetros del argumento del comando PRINT AT. Estas coordenadas coincidirán con X y ABS (COS(X)) respectivamente. El valor de X se modificará, obteniéndose así las diferentes posiciones. Como carácter a imprimir en las respectivas posiciones se utilizará la letra O por su similitud con una pelota. El cuerpo principal del programa lo constituirá, pues, la siguiente instrucción:

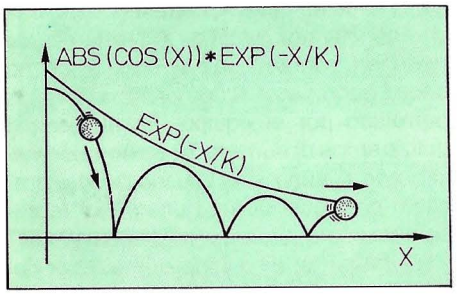
```
30 PRINT AT(X,ABS(COS(X))) "O"
```

Esta línea debe arroparse con un bucle FOR, que permita introducir la variación de X. Puesto que X corresponderá a la coordenada horizontal de AT (columna de texto en pantalla) sus valores límite no deben sobrepasar el número de columnas permitido por el equipo. En el ejemplo utilizaremos el número 30 como valor máximo de X, aunque el usuario puede cambiarlo por aquel que dé una mejor representación en su propia pantalla.
Por otra parte, es sabido que el valor del coseno se mantiene entre 1 y −1. Con lo cual su valor absoluto variará entre 0 y 1. Para conseguir una mayor amplitud en el movimiento vertical será necesario aumentar este rango de variación. Ello se

TABLA DE CONVERSION

ORDENADOR	RAIZ CUADRADA	FUNCIONES TRIGONOMETRICAS					
	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	ASN (<X>)	ACS (<X>)	ATN (<X>)
APPLE II (APPLESOFT)	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
APRICOT (M-BASIC)	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
ATARI	SQR (<X>)	SIN (<X>)	COS (<X>)	—	—	—	ATN (<X>)
CBM 64	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
DRAGON	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
EQUIPOS MSX	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
HP-150	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
IBM PC	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
MPF	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
NCR DM-V (MS-BASIC)	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
NEW BRAIN	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	ASN (<X>)	ACS (<X>)	ATN (<X>)
ORIC	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
SHARP MZ-700 (MZ-BASIC)	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
SINCLAIR QL	SQRT (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	ASIN (<X>)	ACOS (<X>)	ATAN (<X>)
SPECTRAVIDEO	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
ZX-SPECTRUM	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	ASN (<X>)	ACS (<X>)	ATN (<X>)

realiza multiplicando dicha expresión por el número de filas que deba abarcar.



Para que sea posible contemplar la amortiguación de los rebotes de la pelota, habrá que multiplicar la función ABS (COS(X)) por la expresión exponencial EXP (−X/K).

Veinte es un número que dará buen resultado en la mayoría de ordenadores. Al igual que ocurría con el tope de amplitud horizontal, el usuario puede elegir el valor que considere más apropiado para las características de su ordenador.

```
20 FOR X=1 TO 30
30 PRINT AT(X,INT(20*ABS(COS(X))) )"O"
40 NEXT X
```

Como puede observarse, se ha tomado la precaución de calcular la parte entera del resultado de la expresión en la zona "y" del argumento de AT. Ello evitará posibles errores derivados de la exigencia de utilizar un número entero de filas de texto.
Sólo falta poner una línea adicional con una instrucción de borrado de pantalla, para que la trayectoria no quede enmascarada por el texto que pudiera existir en la pantalla.
El programa completo adoptará el siguiente aspecto.

```
10 REM BOLA 1
15 CLS
20 FOR X=1 TO 30
30 PRINT AT(X,INT(20*ABS(COS(X))) )"O"
40 NEXT X
50 END
```

Al ejecutar este primer programa, el resultado no es exactamente el deseado. El problema radica en que el valor del co-

TABLA DE CONVERSION

ORDENADOR	CONVERSION DE ANGULOS		LOGARITMOS		
	DEG	RAD	LOG (<X>)	LOG10 (<X>)	EXP (<X>)
APPLE II (APPLESOFT)	—	—	LOG (<X>)	—	EXP (<X>)
APRICOT (M-BASIC)	—	—	LOG (<X>)	—	EXP (<X>)
ATARI	DEG	RAD	LOG (<X>)	CLOG (<X>)	EXP (<X>)
CBM 64	—	—	LOG (<X>)	—	EXP (<X>)
DRAGON	—	—	LOG (<X>)	—	EXP (<X>)
EQUIPOS MSX	—	—	LOG (<X>)	—	EXP (<X>)
HP-150	—	—	LOG (<X>)	—	EXP (<X>)
IBM PC	—	—	LOG (<X>)	—	EXP (<X>)
MPF	—	—	LOG (<X>)	—	EXP (<X>)
NCR DM-V (MS-BASIC)	—	—	LOG (<X>)	—	EXP (<X>)
NEW BRAIN	DEG (1)	RAD (1)	LOG (<X>)	—	EXP (<X>)
ORIC	—	—	LN (<X>)	LOG (<X>)	EXP (<X>)
SHARP MZ-700 (MZ-BASIC)	—	RAD (<X>) (2)	LN (<X>)	LOG (<X>)	EXP (<X>)
SINCLAIR QL	DEG (<X>) (2)	RAD (<X>) (2)	LN (<X>)	LOG10 (<X>)	—
SPECTRAVIDEO	—	—	LOG (<X>)	—	EXP (<X>)
ZX-SPECTRUM	—	—	LN (<X>)	—	EXP (<X>)

(1) Sólo aplicables en comandos gráficos. (2) Son funciones, devuelven el valor en radianes (RAD) o grados (DEG) del argumento (expresado en la unidad contraria).

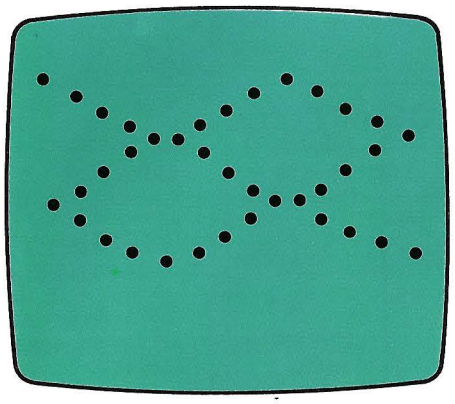
seno se repite cada 6,28 (2x radianes) unidades de X. En consecuencia, su valor absoluto se repite con mayor frecuencia, exactamente cada unidades de X. De esta forma, al variar X de 1 a 30, se efectúan casi 10 rebotes. Ello significa que un rebote se plasma en 3 columnas, con lo que el seguimiento de la trayectoria se hace muy difícil.

La solución consiste en hacer que cada rebote ocupe más espacio. Lo que equivale a ordenar que se produzca un menor número de ellos. La forma de conseguirlo se reduce a hacer que el argumento del coseno varíe dentro de unos límites inferiores. Para ello será suficiente con dividir el argumento del coseno por un número. Cuanto mayor sea este número, menos rebotes se producirán. Dividiendo por 3 la nueva versión del programa será la siguiente:

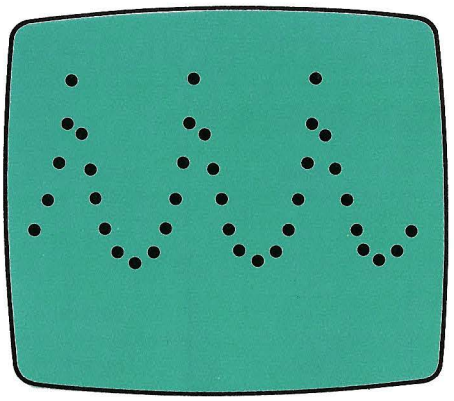
```
10 REM BOLA 2
15 CLS
20 FOR X=1 TO 30
30 PRINT AT(X,INT(20*ABS(COS(X/3))) )"O"
40 NEXT X
50 END
■
```

Esta vez sí que se aprecian los rebotes de la pelota. El único problema es que bota al revés. En efecto, se ve a la pelota caer hacia arriba y rebotar en el límite superior de la pantalla. Este efecto tiene su origen en el hecho de que la coordenada Y del comando PRINT AT aumenta hacia abajo. Es decir: valores mayores proporcionan alturas menores en la pantalla. En el caso que nos ocupa, la expresión situada como coordenada Y proporciona el valor de la altura "aumentando hacia arriba"... Es preciso darle la vuelta. Para ello, bastará con restar la citada expresión de su valor máximo:

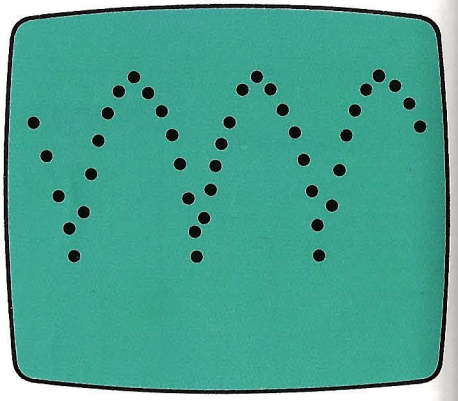
20−INT(20*ABS(COS(X/3))).



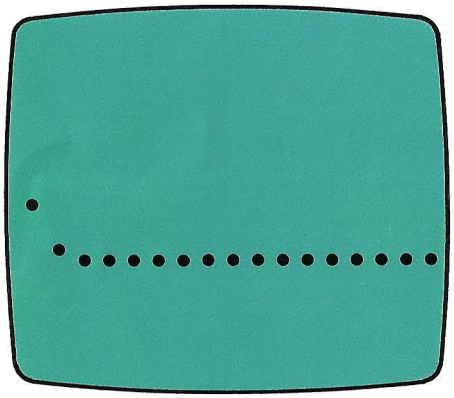
Aspecto de la pantalla al ejecutar el programa BOLA1: no llega a apreciarse con claridad el bote de la pelota.



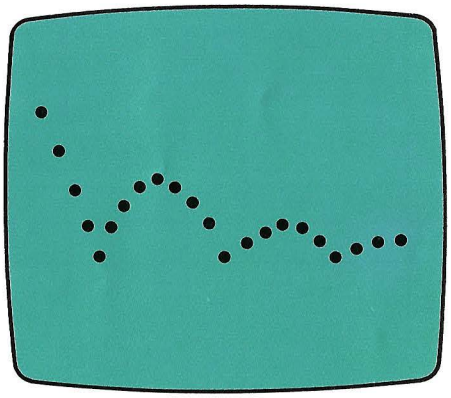
Con el programa BOLA2 es posible ya observar la trayectoria, aunque su trazado aparece invertido.



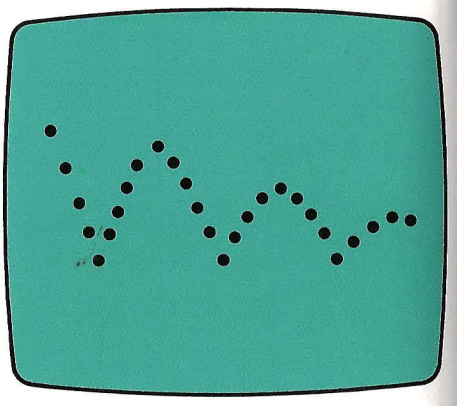
Simulación de la trayectoria de una pelota creada por la ejecución del programa BOLA3.



La ejecución del programa BOLA4 contempla la amortiguación de los sucesivos rebotes. En el caso ilustrado se otorga el valor 1 a la constante K; el resultado es una amortiguación instantánea que no da pie a ningún rebote.



Si la constante K considera igual a 15, la amortiguación resulta ya más realista...



... e incluso puede realzarse asignando un valor superior a K, por ejemplo K=30.

El listado correspondiente a la trayectoria puesta al derecho es el siguiente:

```
10 REM BOLA 3
15 CLS
20 FOR X=1 TO 30
30  PRINT  AT(X,20-INT(20*ABS(COS
(X\3)))"O"
40 NEXT X
50 END
■
```

Aún puede realizarse una nueva mejora Esta consistirá en introducir una amortiguación en los rebotes. Para ello se va a echar mano de otra de las funciones explicadas en este capítulo. La amortiguación a simular será de tipo exponencial, que responde a la siguiente fórmula:

<valor amortiguado>=<valor no amortiguado> * $e^{(-X/K)}$

En ella, X es la variable que irá tomando diferentes valores, mientras que K es una constante que indica la velocidad de amortiguación. En el caso que nos ocupa, dicha fórmula adoptará el siguiente aspecto una vez codificada en BASIC:

```
INT(20*ABS(COS(X/3))) *EXP(-X/K)
```

Aquí, el valor de K ha de ser especificado previamente. Para encontrar el valor ideal puede realizarse un estudio más profundo de la teoría matemática concerniente al hecho. Sin embargo, el camino más corto consiste en utilizar el ordenador para hacer pruebas. En primer lugar se dará el valor 1 a dicha constante para ver qué efecto produce. Así pues, el programa quedará definitivamente como sigue:

```
10 REM BOLA 4
15 CLS
20 FOR X=1 TO 30
30 PRINT AT(X,20-INT(20*ABS(COS(X/3)))
   *EXP(-X/1) )"O"
40 NEXT X
50 END
```

El valor utilizado produce una amortiguación excesiva. La pelota, después de golpear el suelo, no vuelve a ascender. Cambiando el valor de la constante K se conseguirán distintos grados de amortiguación. Algunos de estos valores pueden dar resultados plenamente satisfactorios; al respecto, las figuras muestran trayectorias correspondientes a los valores de K coincidentes con los números 15 y 30.

Pascal (3)

Entrada y salida de datos

La cabecera de un programa PASCAL sirve para dar nombre al programa por medio de un identificador. Este nombre, que sigue a la palabra clave PROGRAM, no tiene un significado especial dentro del contexto del programa; aunque cabe precisar que algunos sistemas particulares pueden asignar a dicho nombre un significado propio fuera del programa para referirse a él específicamente, por ejemplo en orden a su almacenamiento en memorias de masa.

En todo caso, la cabecera del programa tiene otra misión encomendada: declarar una lista de parámetros del programa. La lista de parámetros hace referencia a variables que normalmente serán nombres de ficheros, los cuales deben estar en correspondencia con elementos que ya estén definidos e implantados de alguna forma en el sistema, con idependencia del programa.

Cada ordenador tiene definido un conjunto de caracteres (letras, números y signos especiales), que resultan "inteligibles" para el propio ordenador y para los dispositivos de entrada y salida.

Sobre este conjunto de caracteres, el PASCAL tiene definidas dos variables de tipo fichero: INPUT y OUTPUT. Este tipo de variable se declara con la palabra clave FILE. A pesar de que se estudiará con más detalle posteriormente, conviene adelantar que se trata de un conjunto estructurado de datos en el que todos ellos son del mismo tipo.

El PASCAL tiene ya definido un tipo de variable que se declara con la palabra reservada TEXT y que no es más que un fichero de caracteres (FILE OF CHAR). Los ficheros estándar INPUT y OUTPUT son de este tipo, de forma que, para el programa PASCAL, es como si previamente se hubiera realizado la siguiente declaración:

```
VAR INPUT,OUTPUT: TEXT;
```

En consecuencia, al escribir un programa no será necesario volver a declarar estas variables, sino únicamente mencionarlas en la lista de parámetros de la cabecera.

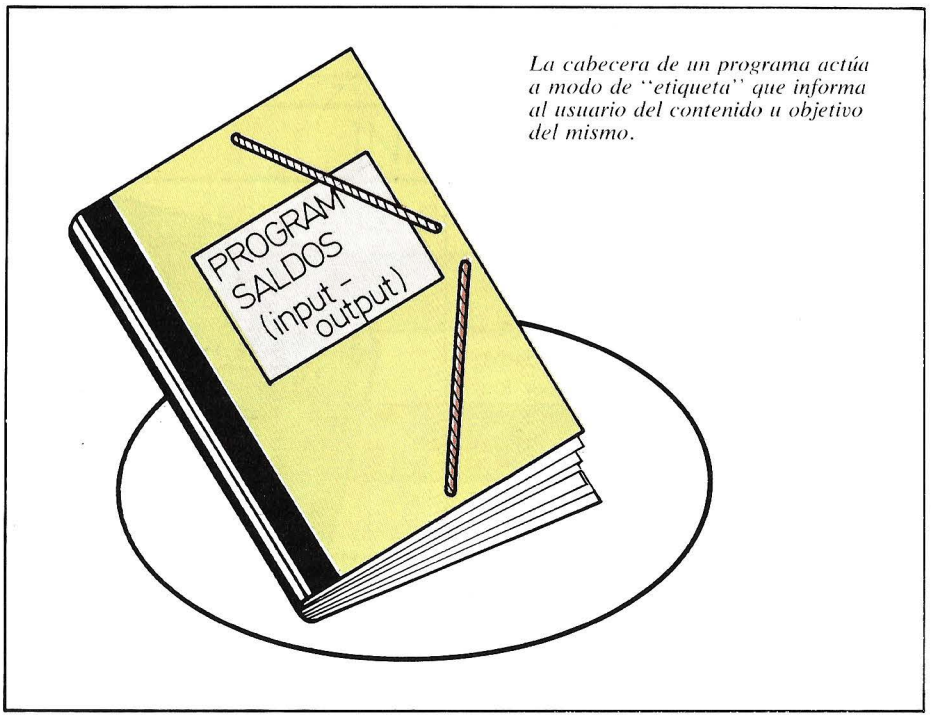
INPUT y OUTPUT, como parámetros formales del programa, representan a los dispositivos de entrada/salida de un sistema informático: el teclado, la impresora o la unidad de discos. Gracias a ellos se facilita la comunicación entre un programa PASCAL y el sistema bajo cuyo control se esté ejecutando el programa.

Como los ficheros INPUT y OUTPUT son utilizados muy a menudo por la mayoría de los programas, se consideran como parámetros implícitos en ciertas funciones y procedimientos estandarizados cuando no se especifica lo contrario. Por lo tanto, serán inicializados automáticamente y con antelación a la ejecución del programa, al ser declarados en la cabecera del mismo.

Si en un programa se va a hacer referencia a otros ficheros externos —ya existentes antes de la ejecución del programa, o que van a ser creados durante la ejecución—, los nombres de estos ficheros deben aparecer en la lista de parámetros del programa; además, deben ser declarados en la zona de definiciones y declaraciones como variables de tipo fichero. Por ejemplo:

```
PROGRAM EJEMPLO (INPUT,OUTPUT,
X,ARCHIVO); VAR X,ARCHIVO: FILE
OF <tipo>;
```

siendo <tipo> el tipo de las variables que constituyen ese fichero.



La cabecera de un programa actúa a modo de "etiqueta" que informa al usuario del contenido u objetivo del mismo.

LECTURA DE DATOS

Un programa sin intercambio de información con el exterior tendrá poca utilidad al realizar siempre la misma función. En cambio, al poder recibir información del exterior ve aumentada su flexibilidad: proporcionará resultados de salida diferentes cada vez, en función de los datos que reciba. Suponiendo, por ejemplo, que ya se tienen declaradas con anterioridad las variables tensión, resistencia y corriente, como reales, las siguientes sentencias de asignación calcularán un valor para la variable tensión aplicando la ley de Ohm:

```
resistencia:=100;
corriente:=3;
tensión:=resistencia*corriente;
```

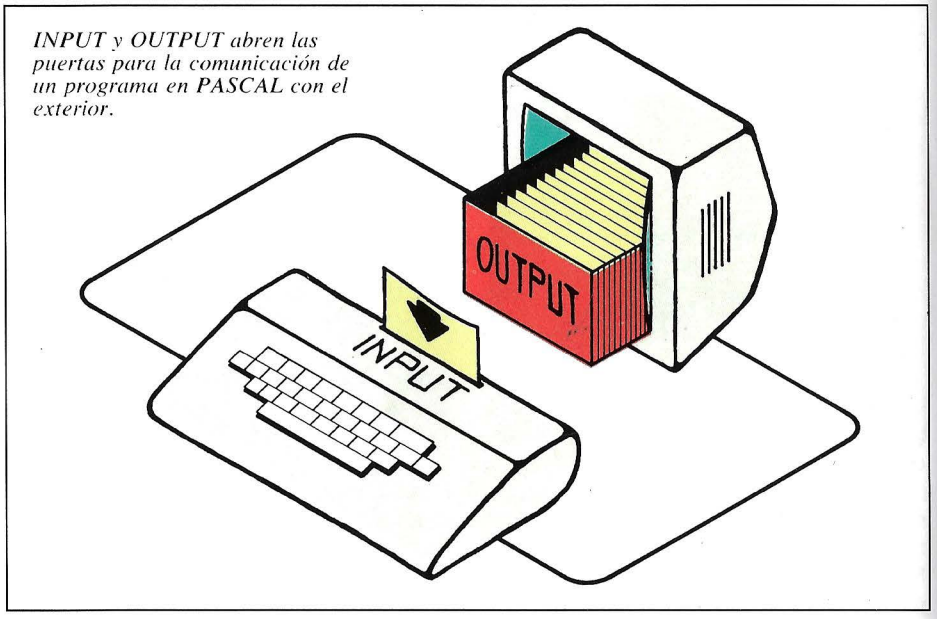
No obstante, para calcular ahora la tensión si la corriente vale por ejemplo 2, será preciso modificar el programa, incluyendo una nueva sentencia de asignación: corriente:=2; en lugar de la ya existente, lo que resta efectividad al programa. Para resolver esta situación, hay que utilizar una o más sentencias de entrada de datos, que en PASCAL se codifican mediante el comando READ. Esta sentencia lee un dato del fichero de entrada, especificado en el argumento de la propia sentencia de lectura, y lo asigna a la variable del programa que se incluye a continuación, en la misma sentencia READ.
Si no se especifica la fuente de los datos, se asume como tal al fichero estándar INPUT. En consecuencia, cuando se utiliza este fichero, no es necesario indicarlo expresamente en la sentencia READ. El formato genérico de la sentencia es el siguiente:

READ(<fichero>,<var.>,<var.>,...);

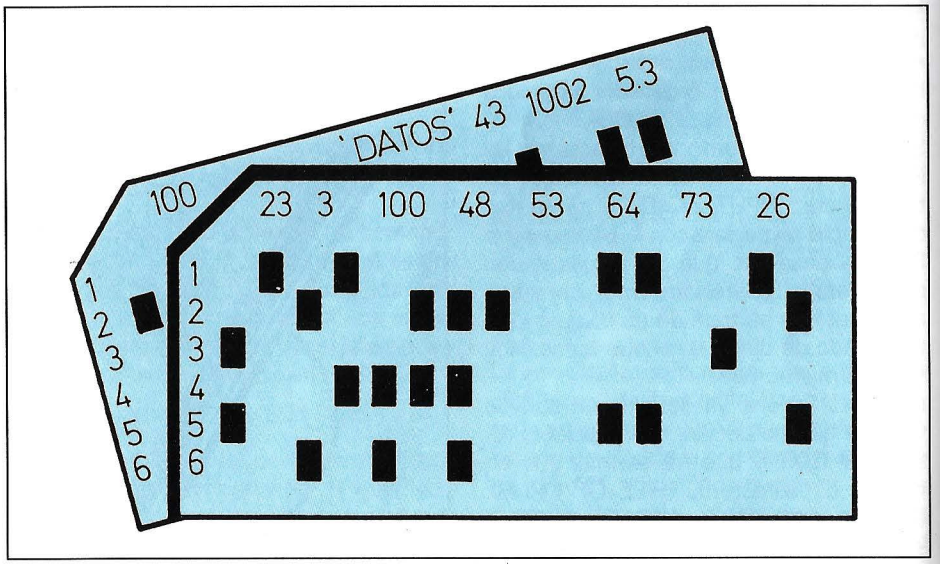
El campo <fichero> es opcional y coincide con el nombre de la fuente de datos de entrada de la que se van a leer los valores. A su vez, <var> son las variables que recibirán la asignación de los datos leídos.
Así pues, se puede leer un solo dato del fichero estándar INPUT, aunque también se pueden leer varios datos de ese mismo fichero y asignarlos al grupo de variables, separadas por comas que figuren en la sentencia READ; o bien, es posible, asimismo, extraer uno o varios datos para asignarlos a las variables desde cualquier otro fichero externo, que previamente se haya especificado en la lista de parámetros de la cabecera como fuente de los datos de entrada.
Para aumentar la utilidad del ejemplo anterior, pueden sustituirse las dos primeras sentencias de asignación por la siguiente instrucción READ:

READ(resistencia,corriente);

Con lo que el programa leerá los datos de resistencia y corriente y obtendrá un resultado dependiente de los datos leídos. En el soporte de entrada, los datos deben aparecer separados por espacios en blanco o por delimitadores de fin de línea; ello difiere, normalmente, para cada compilador PASCAL específico y dependerá del tipo de entrada de datos, esto es: si tiene lugar a través del teclado o por medio de cualquier otro dispositivo como, por ejemplo, un fichero de entrada residente en cinta o en disco magnético.
Al utilizar la sentencia READ, los datos a leer estarán dispuestos en líneas; de tal forma que cuando se hayan leído todos los datos de una línea, se saltará automáticamente a la línea siguiente. La lectura de los mismos se realizará siempre de izquierda a derecha.



INPUT y OUTPUT abren las puertas para la comunicación de un programa en PASCAL con el exterior.



La posibilidad de introducir datos otorga una gran versatilidad a la máquina: permite la elaboración de distintos datos y la consecución de diversos resultados a partir de un mismo programa.

Si se quiere forzar el salto a una nueva línea de entrada, se puede utilizar la sentencia:

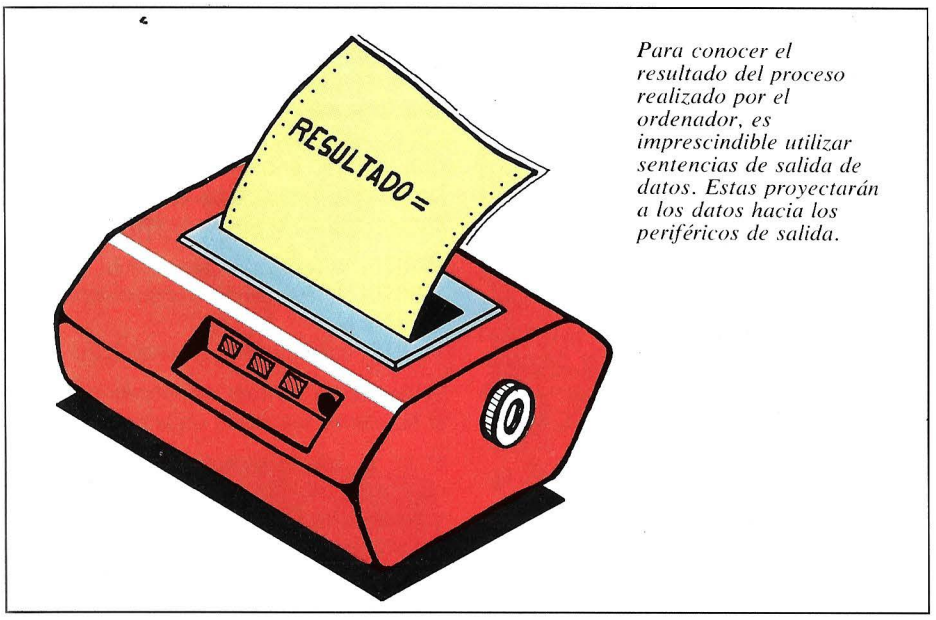
READLN(<fichero>,<var.>,...);

Tal como se observa, su formato es análogo al de la sentencia READ convencional, con la única diferencia de que ahora, después de leer las variables especificadas, se salta a la siguiente línea de entrada, aunque aún no se hayan terminado de leer todos los datos de esa línea. Los datos que sigan al último leído serán ignorados, y la próxima sentencia de lectura leerá el primer dato de la línea siguiente. Así, por ejemplo, si los datos de entrada están distribuidos en dos líneas de la siguiente forma:

```
100  25
2    50
```

la ejecución de las sentencias que siguen asignará el valor 100 a resistencia y saltará a la siguiente línea, ignorando el dato 25, para asignar el valor 2 a la variable corriente:

```
READLN(resistencia);
READLN(corriente);
```



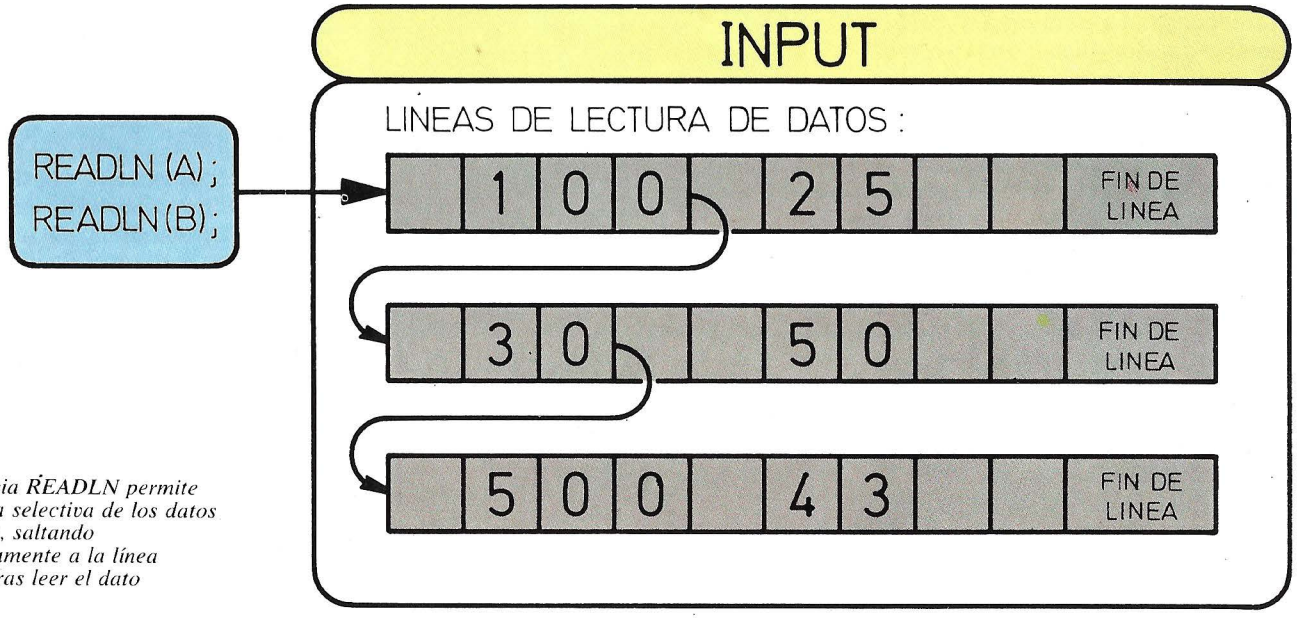
Para conocer el resultado del proceso realizado por el ordenador, es imprescindible utilizar sentencias de salida de datos. Estas proyectarán a los datos hacia los periféricos de salida.

SALIDA DE DATOS

El resultado de un programa no suele resultar útil si no es comunicado al exterior a través de algún dispositivo de salida asociada al ordenador. La misión de una instrucción de salida es la de presentar cualquier expresión o valor de una variable en la pantalla o en cualquier otro dispositivo de salida como, por ejemplo, la impresora. En PASCAL, la sentencia especializada en este cometido es WRITE, cuyo formato coincide con:

WRITE([<fichero>,]<exp.>[,<exp.>,...]);

El campo <fichero> define al fichero de salida de los datos y su presencia es opcional; si bien, en el caso de incluirse, resultará imprescindible haberlo mencionado en la lista de parámetros de la cabecera, tal y como ocurría con la sentencia READ. La zona <exp.> contiene a las expresiones matemáticas o, simplemente, a las variables cuyo valor se desea representar. En este caso, cuando no se especifique el fichero de destino de los datos de salida, se asumirá como tal al fichero estándar OUTPUT. Continuando con el ejemplo inicial, para que sea posible conocer cuál ha sido el resultado de la ejecución, habrá que añadir una sentencia de salida. El programa completo adoptará el siguiente aspecto:



La sentencia READLN permite una lectura selectiva de los datos de entrada, saltando automáticamente a la línea siguiente tras leer el dato ordenado.

TABLA DE COMANDOS-PASCAL

Comando	Cometido	Observaciones
INPUT	Declarar fichero estándar de entrada	Cabecera del programa
OUTPUT	Declarar fichero estándar de salida	Cabecera del programa
TEXT	Declaración de tipo fichero de caracteres	Declaraciones y definiciones
READ({<fich.>} <var.>{,...});	Lectura de datos de un fichero	Bloque de sentencias
READLN({<fich.>} <var.>{,...});	Lectura de datos y cambio de línea	Bloque de sentencias
WRITE ({<fich.>} <exp.>{,...});	Escritura de expresiones	Bloque de sentencias
WRITELN ({<fich.>} <exp.>{,...});	Escritura de expresiones y cambio de línea	Bloque de sentencias

```
PROGRAM LEYDEOHM (INPUT, OUTPUT);
VAR tensión, resistencia, corriente: REAL;
BEGIN
   READ(resistencia,corriente);
   tensión:=resistencia*corriente;
   WRITE(tensión)
END.
```

Como en el caso de READ, en una sentencia WRITE se pueden incluir varios elementos para ser presentados; éstos deben aparecer separados por comas. Si los referidos elementos o parámetros son expresiones matemáticas, se evaluarán previamente, y el valor obtenido será presentado a través del dispositivo de salida que corresponda. La sentencia WRITE del ejemplo se podría haber escrito como:

```
WRITE(resistencia*corriente);
```

De esta forma, se evaluaría el producto antes de mostrar su valor resultante. Tal y como aparece, el programa ejemplo resulta bastante frío. Si la entrada se realiza a través del teclado, puede realizarse su presentación externa ordenando al programa que indique con un mensaje qué tipo de entrada es la que espera recibir. Para ello se puede utilizar la sentencia. WRITE ('introducir resistencia y corriente'), precediendo a la instrucción READ, con lo que el literal del paréntesis se reproducirá tal cual en la pantalla.
Cada expresión de la sentencia WRITE puede, opcionalmente, adoptar la forma:

<exp.>:<C1>:<C2>

La zona <C1> es un número natural que indica el número de caracteres con que debe ser escrita la expresión <exp.>. En el caso de que <exp.> necesite menos caracteres, se completará la expresión con espacios en blanco por la derecha. <C2> es otro número natural que revela el número de dígitos de que constará la parte fraccionaria de <exp.> en el caso de que ésta sea de tipo REAL. Así, por ejemplo, las siguientes líneas:

```
READ(tensión,resistencia);
corriente:=tensión/resistencia;
WRITE('corriente=',corriente:7:3);
```

producirán la siguiente salida:

```
corriente=   1.818
```

desde luego, suponiendo que el valor de la tensión es 200 y el de la resistencia 110.
Como se observa, la sentencia WRITE presenta los datos en una única línea. Si se desea presentarlos en varias líneas será preciso utilizar una sentencia WRITELN. Esta comparte el mismo formato que WRITE, si bien, después de escribir las expresiones que se le indiquen, provocará un salto a la siguiente línea de salida:

```
WRITELN('tensión=',tensión:4);
WRITELN('resistencia=',resistencia:4);
WRITELN;
WRITE('corriente=',corriente:7:3);
```

El bloque de sentencias anterior producirá el siguiente resultado en la pantalla:

```
tensión=  200
resistencia=  110

corriente=    1,818
```

En efecto, se observa que la sentencia WRITELN sin parámetros genera como resultado la escritura de una línea en blanco.

```
WRITELN (TENSION);
WRITELN (RESISTENCIA);
WRITELN;
WRITELN (CORRIENTE);
```

```
TENSION = 200
RESISTENCIA = 110

CORRIENTE = 1,818
■
```

Cuando la salida de datos se ordena a través de una sentencia WRITELN, se producirá un salto automático a la próxima línea de presentación tras visualizar el dato o mensaje.

MS/DOS (5)

Editor y Debugger

Cuando el usuario se dispone a preparar sus programas, generar ficheros de datos o de texto, o cualquier otra tarea relacionada con el mantenimiento intrínseco de la información contenida en los ficheros, se ve obligado a recurrir al denominado editor.

EL EDITOR

Tal es el nombre que recibe la herramienta básica para este tipo de operaciones. En el mercado actual existen un gran número de ellos, cada uno con distintas características y posibilidades. Algunos, como los procesadores de textos, son editores de gran potencia, si bien resultan excesivos para tareas tales como el mantenimiento de programas o de simples ficheros de texto; de ahí que para el cometido señalado se emplee el editor que viene asociado al sistema operativo.

El editor del MS/DOS se presenta a modo de comando externo, invocable por medio de la orden EDLIN. Concretamente se trata de un editor de líneas, de ahí su denominación (EDitor LINe), por contraste con otros editores denominados de pantalla completa o "full screen", los cuales operan sobre el contenido total de la información visualizada en cada instante en la pantalla. Ello no implica, sin embargo, que este editor sea de peor calidad sino, sencillamente, que está orientado a trabajar línea a línea.

Una característica importante del editor del MS/DOS es que comparte su método de operación, por lo que al almacenamiento de información se refiere, con el editor del sistema operativo CP-M. Así, por ejemplo, no elimina la versión antigua de un fichero, sino que la almacena con el mismo nombre, aunque caracterizándola con la extensión "BAK". A su vez, la nueva versión toma el nombre original del fichero fuente. Ello obliga al usuario a tomar las adecuadas precauciones con los nombres de los diferentes ficheros, con objeto de que la distinción entre ellos no se reduzca exclusivamente a la extensión del nombre.

Semejante procedimiento tiene el inconveniente de que se duplica la información almacenada y si se editan muchos ficheros puede llegar a ocuparse con rapidez el soporte de almacenamiento.

- *Operación con el editor*

Cuando se invoca al editor, hay que especificar sobre qué fichero se va a actuar, siguiendo el procedimiento que se indica a continuación:

EDLIN A:DATOS.TEXT

Tras esta operación, el editor inicia un proceso de búsqueda y carga en memoria del fichero solicitado, brindando al usuario distintas respuestas de acuerdo a la situación que se plantee:

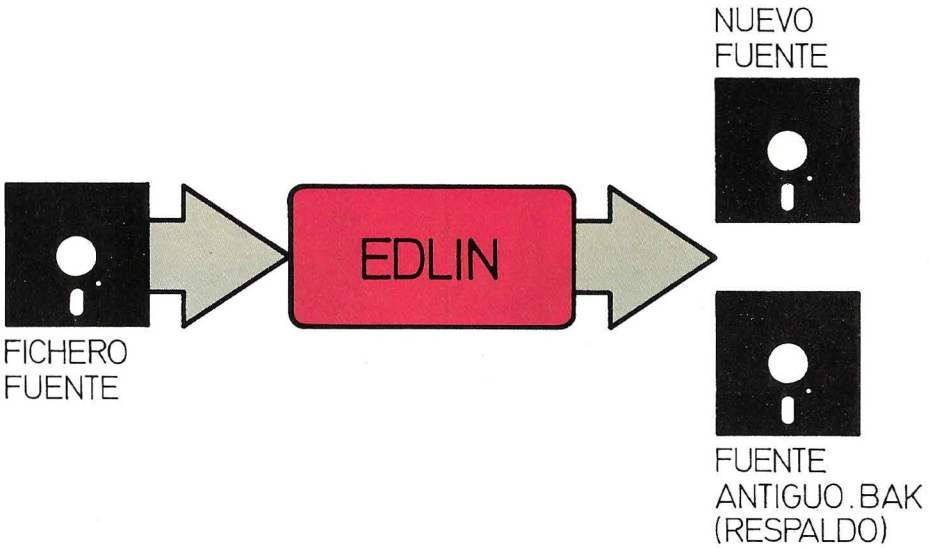
a) Si el fichero no resulta localizable, bien porque sea nuevo y deseemos crearlo, o bien porque se haya introducido un nombre erróneo, aparecerá en la pantalla el mensaje de "nuevo fichero":

New File

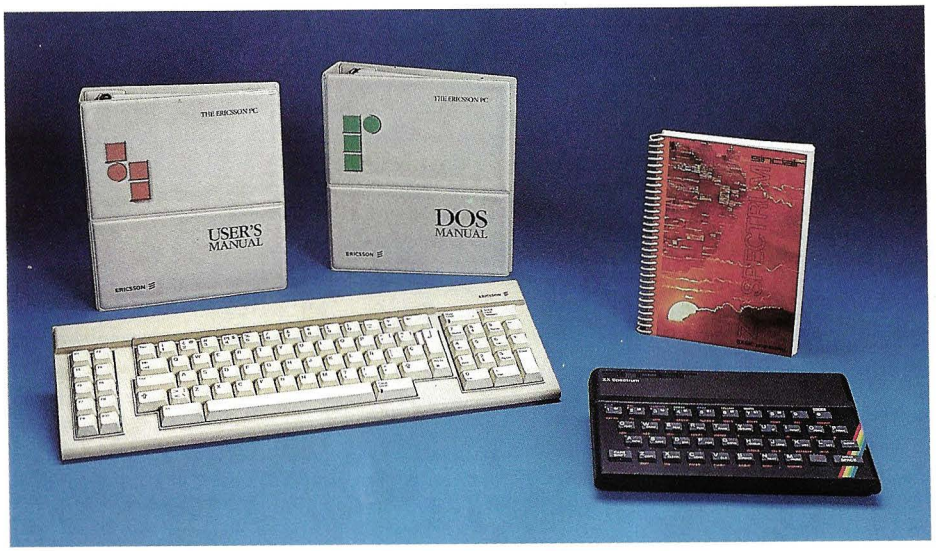
b) Si se encuentra el fichero en cuestión, y su tamaño es adecuado para la carga de memoria y posterior operación con el mismo, la pantalla ofrecerá el mensaje:

End of Input File

c) Puede ocurrir que el fichero llegue a



Tras realizar su función, el editor del MS/DOS conserva siempre una copia de seguridad o de respaldo (backup) de la información previa.



La presencia de un sistema operativo, dotado de potentes comandos y herramientas de edición y puesta a punto de programas, contribuye a distanciar a los equipos profesionales de los ordenadores domésticos.

localizarse pero que su tamaño sea superior al de la memoria disponible. En tal caso no aparecerá mensaje alguno y el editor no cargará en memoria todo el fichero, sino tan sólo una parte del mismo, procediendo a actualizaciones sucesivas de su contenido.

Tras este proceso, el editor queda listo para operar, cosa que indica presentando su propio "prompt": un asterisco (*). A partir de ahí, el usuario tiene a su disposición el conjunto de comandos del editor, entre los que se señalan a continuación los más importantes.

El comando E se encarga de dar por terminada la tarea de edición, y activa el proceso de creación del nuevo fichero fuente y del fichero de respaldo. El comando Q también finaliza la edición, si bien el resultado no afecta a los ficheros fuente y de respaldo; por lo cual, el trabajo realizado no se refleja en el fichero cuyo contenido se ha editado.

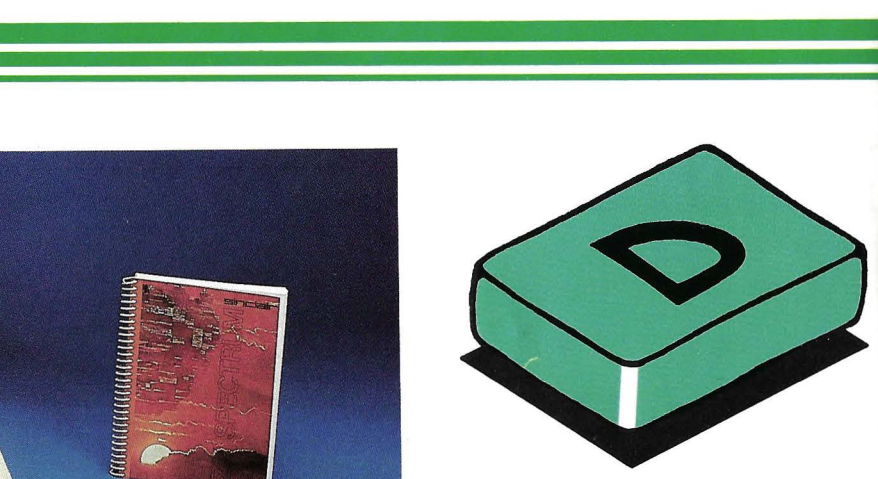
Las funciones de desplazamiento a través del fichero y de inserción y borrado de la información, se realizan por medio de tres comandos básicos: I (Insertar), D (Borrar) y L (Listar). Una vez seleccionada la línea a partir de la que se desea introducir información, la activación del comando I permitirá la inclusión de sucesivas líneas de datos o de programa. Cada vez que se introduce una nueva línea y se pulsa la tecla RETURN o ENTER, el sistema la almacena en el buffer del editor y queda en disposición de aceptar otra línea.

Cuando se desea crear un fichero nuevo, éste será el primer comando que debe entrar en acción. Al finalizar la entrada de nueva información y con el fin de regresar al estado normal de operación del editor, es necesario pulsar la tecla BREAK o <crtl −C>. Hecho esto, aparece de nuevo el prompt del editor (el asterisco).

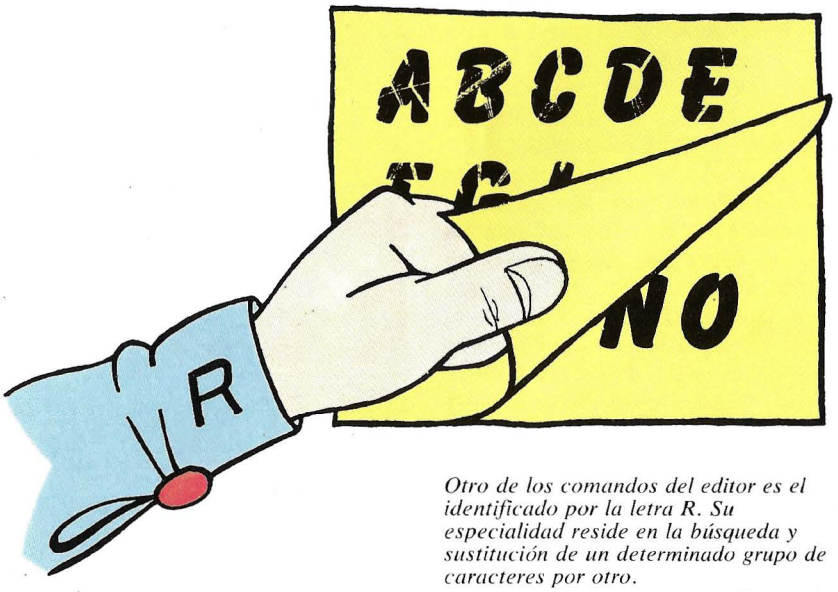
El comando D resulta adecuado para borrar a voluntad una o varias líneas de información. La forma que puede tomar esta orden es variable, aunque las más importantes son las que se relatan a continuación. Si se pulsa D sin añadir cualquier otro parámetro, será borrada la línea con la que se opera en ese preciso instante. Por el contrario, si se especifica un número de línea, será ésta la afectada por el borrado. Para eliminar un bloque de líneas de una sola vez, hay que optar por uno de los dos procedimientos que siguen. El primero adopta la formulación:

<n' de línea final> D

En este caso, son borradas todas las lí-



El editor del MS/DOS es de los denominados "editores de líneas": su trabajo se concreta sobre una sola línea simultáneamente. Entre los comandos de edición se encuentra el de borrado, invocable por medio de la letra D.



Otro de los comandos del editor es el identificado por la letra R. Su especialidad reside en la búsqueda y sustitución de un determinado grupo de caracteres por otro.

neas comprendidas entre la línea actual de trabajo y la señalada como línea final. El número de la línea inicial se toma por defecto.

El segundo método obedece a la expresión:

<n' línea Inicial>,<n' línea final> D

Ahora las líneas eliminadas son las delimitadas por los números señalados en el comando.
El comando L permite al usuario desplazarse de un punto a otro del fichero en edición para ver su contenido y modificarlo si fuera necesario. Su modo de operación es similar al propio del comando Delete (D).
Si se invoca simplemente pulsando la tecla L, aparecerán en la pantalla un total de 23 líneas de texto delimitadas por la línea en curso de edición; esto es: localizadas por encima o por debajo de ésta. No obstante, de especificar un número de línea en la forma:

<n' Línea> L

se visualizarán las 23 líneas, si bien, la central corresponderá con la línea especificada. Esta última formulación es la que permite desplazarse al punto del fichero que interese visualizar; desde luego, éste punto no tiene por qué coincidir con la línea actual en operación. También es posible por medio de este comando visualizar un determinado bloque de líneas, precisando los números de las líneas inicial y final del mismo:

<n' línea inicial>,<n' línea final> L

Por último, cabe mencionar a dos comandos de gran interés para el usuario, ya que permiten la localización rápida de determinados conjuntos de caracteres a los que el programador o el usuario desea acceder directamente, para examinarlos o modificarlos. Estos comandos responden a los apelativos S (búsqueda) y R (sustitución).

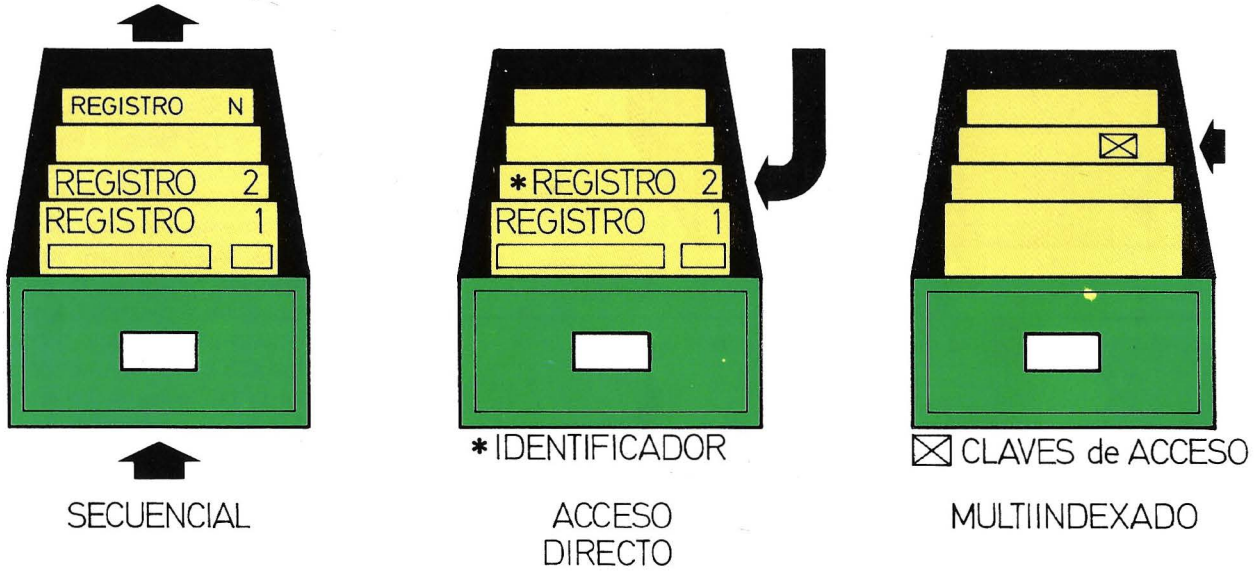
Ficheros para el almacenamiento de datos

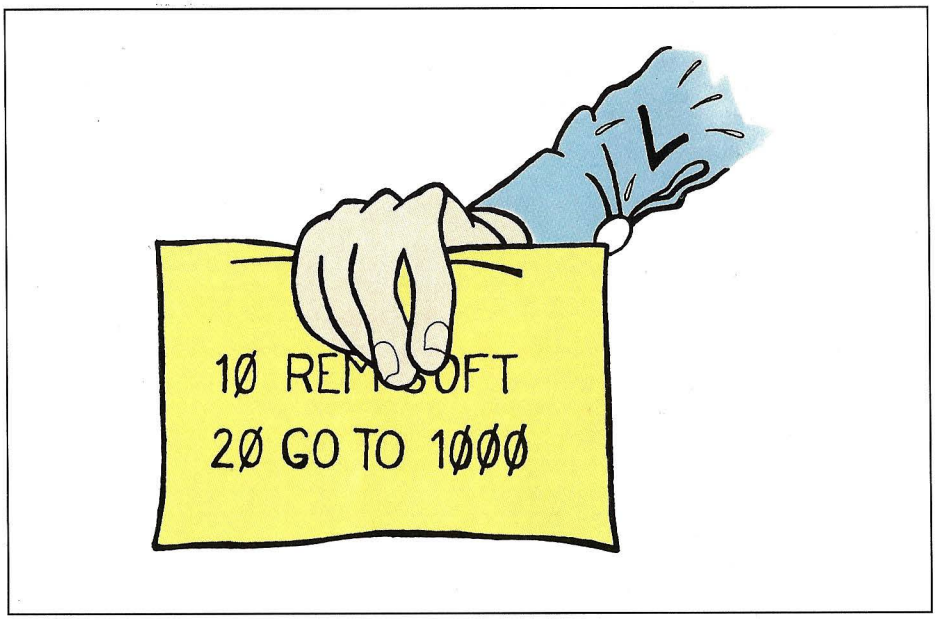
Una de las bases de la informática reside en la posibilidad de almacenar información, de tal manera que pueda ser posteriormente recuperada y utilizada en otros procesos. El almacenamiento afecta tanto a los programas cuya misión es procesar los datos, como a los propios datos manipulados.
Un *fichero* de datos está constituido por un conjunto de *registros,* los cuales acogen los datos distribuyéndolos internamente en una serie de subdivisiones denominadas *campos*. Como ejemplo, cabe imaginar a un registro como la porción del archivo que almacena el conjunto de datos correspondientes a una persona: su nombre, apellidos y dirección; mientras que los campos serían las porciones del registro que contienen los valores asociados a cada uno de los tres conceptos.

Los ficheros pueden clasificarse en muy diversas categorías, entre las que cabe destacar a los ficheros de tipo secuencial, de acceso directo y multi-indexados.
En síntesis, un fichero es de tipo secuencial cuando para acceder a la información contenida en uno cualquiera de sus registros es necesario ir leyendo, uno tras otro y desde el principio, hasta llegar al registro que contiene la información buscada.

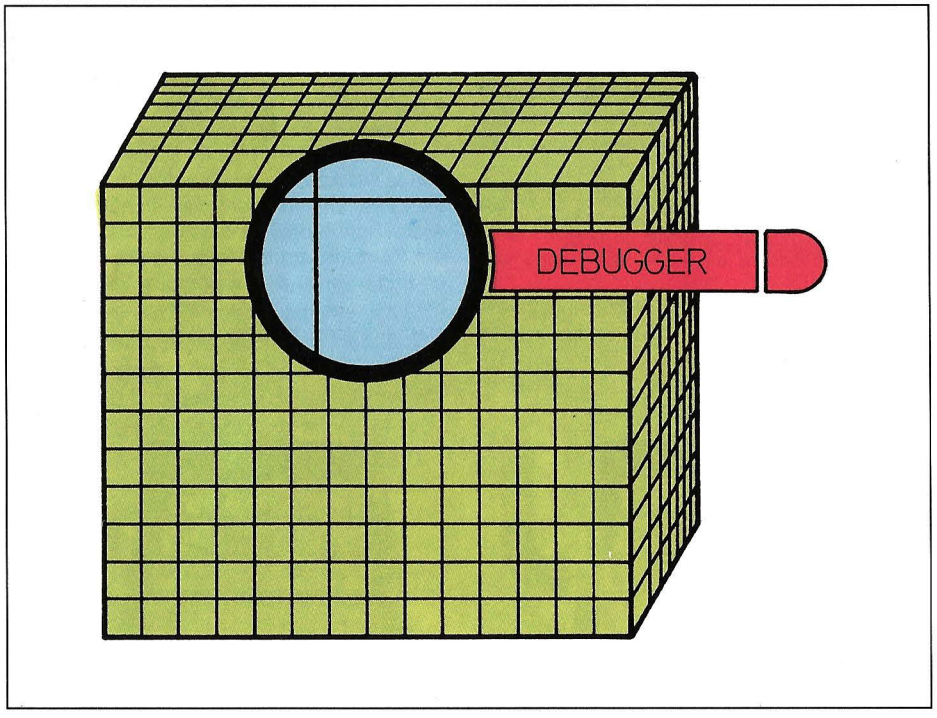
Obviamente, este método no contribuye en absoluto a favorecer el rápido acceso a la información.

En un fichero de acceso directo, cada registro tiene asociado un número de identificación, y el acceso a la información en el contenido se efectúa a través del número que identifica al registro. Este método de acceso resulta muy conveniente cuando no se accede a más del 20% de los registros del fichero. Los ficheros multi-indexados funcionan de forma muy similar a los de acceso directo, con la salvedad de que, en este caso, el elemento identificador no es un número exclusivo y propio de cada registro, sino que coincide con la información contenida en uno a varios de sus campos. Estos campos que se utilizan como identificadores son los que se denominan *claves*.





Cuando el programador desee visualizar una línea o un conjunto de líneas del fichero en edición, debe recurrir al comando L en una de sus múltiples formulaciones.



El "debugger" es una herramienta del sistema operativo MS/DOS orientada a la corrección y puesta a punto de programas.

El comando S se utiliza para ordenar la búsqueda de un determinado grupo de caracteres en el fichero; por ejemplo, una determinada palabra. El formato más general del referido comando es el siguiente:

<n' línea inicial><n' línea final>S
<caracteres a localizar>

Bajo este formato se producirá la búsqueda de los caracteres a localizar entre las líneas señaladas. Cada vez que se detecte su presencia, el ordenador mostrará la línea en la que se encuentran. Tras iniciar la búsqueda, bastará con pulsar S, sin introducir ningún tipo de parámetro adicional, para que ésta prosiga automáticamente saltando, de grupo de caracteres en grupo de caracteres.

El comando R opera de forma análoga, con la única diferencia de que hay que especificar otro grupo adicional de caracteres, el cual sustituirá al inicialmente señalado como objetivo de la búsqueda. Una vez iniciado el proceso, cada vez que se localice el referido grupo de caracteres se interrumpirá la búsqueda y la máquina interrogará al usuario para conocer si éste desea que los caracteres sean sustituidos por los nuevos o no.

DEBUGGER

El "debugger" es una poderosa herramienta que el MS/DOS pone a disposición del usuario especializado para la corrección y puesta a punto de sus programas. Como tal, está destinada al programador y no al usuario cuyo único deseo es la explotación de paquetes de aplicación ya ultimados. Con la ayuda del "debugger", es posible acceder a las distintas posiciones de memoria del ordenador, consultar su contenido y, de esta forma, comprobar si los resultados intermedios del programa son correctos; en el caso de que estos no sean correctos, el programador estará en condiciones de precisar a partir de qué punto empiezan a producirse los fallos, lo que facilitará en gran medida su corrección.

Los detalles operativos del "debugger" son sumamente dependientes del microprocesador que constituye el cerebro de cada máquina, de ahí que no entremos en detalles sobre el mismo. No obstante, sí es oportuno precisar que el "debugger" del MS/DOS opera con aritmética hexadecimal y, por ello, el contenido de las posiciones de memoria examinadas puede aparecer ante el usuario bajo un aspecto críptico. Hay que tener en cuenta, además, que el "debugger" revelará, por medio de abreviaturas, cuál es la naturaleza o tipo de la información almacenada en cada posición de memoria.

Microsoft Chart (1)

Un paquete horizontal para gráficos de gestión

La compañía americana MICROSOFT es la fuente de diversos programas de aplicación estandarizados, entre los que cabe destacar el procesador de textos WORD, la hoja electrónica MULTIPLAN y el paquete para gráficos de gestión MICROSOFT CHART. En el presente capítulo, da comienzo el estudio de este último producto.

El paquete gráfico de MICROSOFT puede utilizarse de forma coordinada con otras aplicaciones de la misma compañía; así, por ejemplo, se pueden introducir los datos mediante la hoja electrónica MULTIPLAN y, a continuación, sin necesidad de realizar ningún proceso adicional, utilizar esos mismos datos para obtener representaciones gráficas. No obstante, también puede utilizarse el programa MICROSOFT CHART de forma autónoma, realizando la entrada de datos dentro de su entorno.

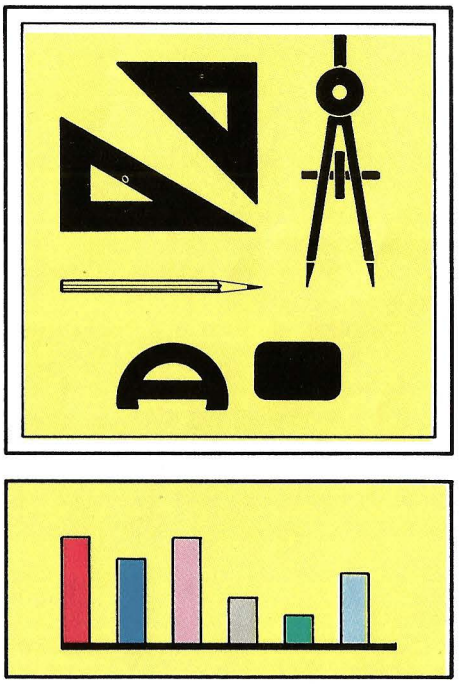
Esta aplicación gráfica está disponible en distintas versiones, adecuadas para trabajar con los principales sistemas operativos: CP/M, MS/DOS, APPLE-DOS, APPLE MACINTOSH... Ello significa que el paquete de MICROSOFT CHART puede ser explotado en prácticamente cualquier ordenador personal.

"FILOSOFIA" DEL MICROSOFT CHART

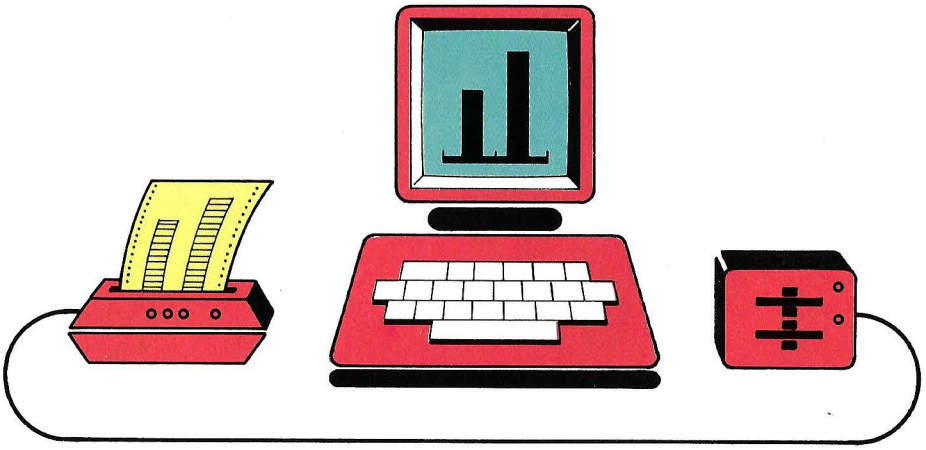
Como se mencionó en el capítulo de introducción al software gráfico, los paquetes de gestión tienen como principal objetivo proyectar un conjunto de datos numéricos (alfabéticos sólo en algunos casos muy determinados) en un diagrama que ofrezca una visión más fácilmente interpretable. El MICROSOFT CHART no es una excepción.

Su metodología de trabajo resulta muy sencilla y eficiente. En primer lugar, admite pares de valores (por defecto toman como nombre X e Y) que pueden representar cualquier fenómeno. En este punto hay que señalar que la terminología MICROSOFT utiliza como descripción Categoría y Valor (Category y Value) para X e Y respectivamente. Después de haber introducido los datos, el usuario puede elegir entre varias alternativas de representación gráfica (barras, líneas, tartas...) para que, de forma automática, aparezca en la pantalla el diagrama correspondiente. En algunos casos está permitido incluso superponer dos tipos de diagramas; por ejemplo, los mismos datos se pueden representar en un diagrama de barras y "encima" de las barras en forma de líneas.

Además de esta propiedad, que se puede considerar de forma global para todos los programas gráficos de gestión, MICROSOFT CHART también permite una cómoda edición de datos. Al efecto posee



Los tradicionales útiles para la elaboración de gráficos tienden a convertirse en obsoletos ante el constante empuje de las aplicaciones gráficas para ordenador.



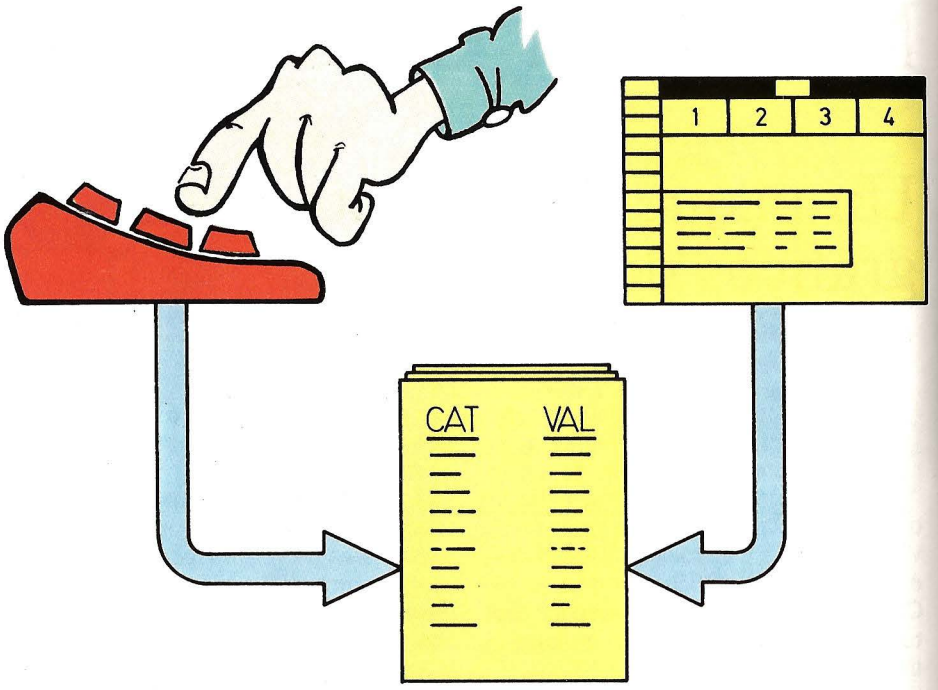
Hoy en día, con un pequeño ordenador personal y un paquete para confeccionar gráficos de gestión, queda resuelta cualquier necesidad de esta índole con rapidez, comodidad y perfección en el resultado.

una serie de formatos estandarizados para fechas, porcentajes, centrados, etc., que pueden ser sustituidos por formatos propios del usuario. Otra de sus posibilidades de gran interés es la de facilitar la inclusión de leyendas dentro del gráfico.
Los próximos apartados se encargarán de describir cada una de estas posibilidades que, en el fondo, constituyen los únicos factores que permite diferenciar entre sí a los distintos programas gráficos existentes en el mercado.

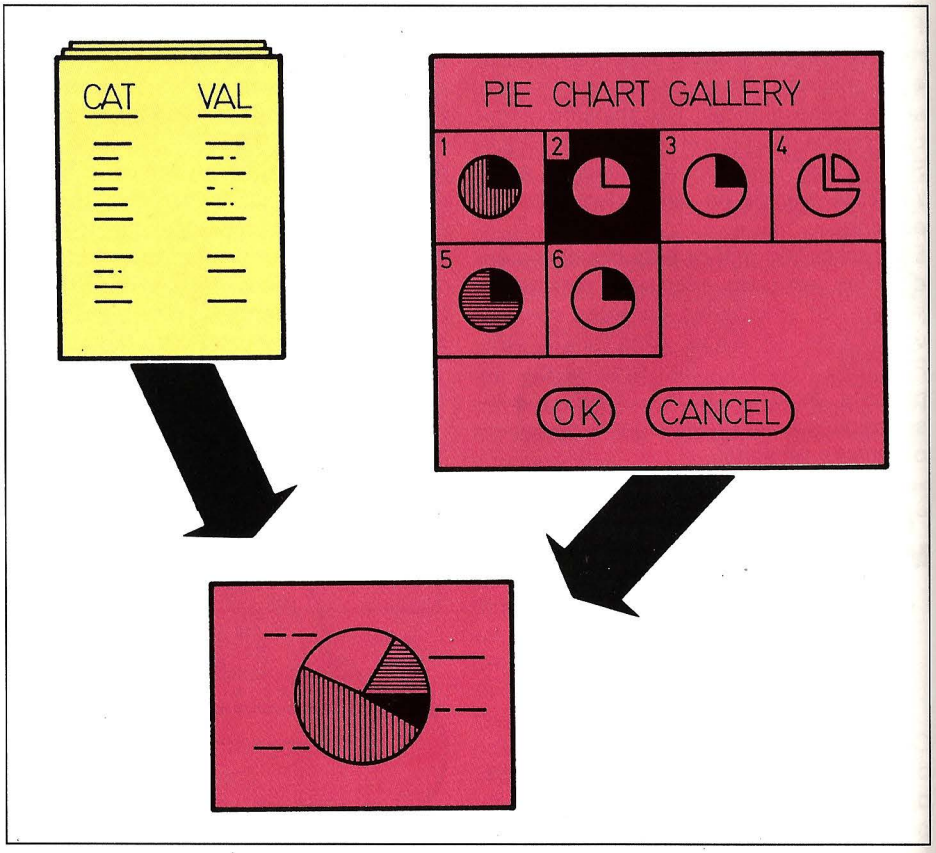
CARACTERISTICAS DEL MICROSOFT CHART

Para iniciar una sesión de trabajo con esta aplicación hay que poner en práctica los tres pasos tradicionales para arrancar cualquier programa:
1. Conectar el ordenador e inicializarlo con el sistema operativo.
2. Introducir el disco que contiene el programa en la correspondiente unidad.
3. Activar la aplicación cargando el programa en la memoria principal.
Tras estos pasos rutinarios, aparecerá en la pantalla el menú principal del programa. En el mismo figuran seis opciones entre las que el usuario puede elegir:

1. Comandos generales del sistema operativo.
2. FILE: opción adecuada para el almacenamiento de información en disco y, en general, para realizar las operaciones de control de la memoria auxiliar.
3. EDIT: su misión es realizar las típicas labores de un editor; sirve para incluir datos y comentarios en las etiquetas de los diagramas.
4. DATA: si se elige esta opción, el programa permitirá la creación y/o modificación de los datos que se utilizan para producir los gráficos.
5. GALLERY: cuando el usuario tiene introducidos los datos, debe elegir el tipo de diagrama en el que desea que aparezca la representación. La opción GALLERY sirve precisamente para ello, ofreciendo una serie de formatos gráficos entre los que el usuario puede elegir.
6. CHART: se puede utilizar para producir la representación completa del gráfico.
7. FORMAT: adecuada para resolver todos los problemas de formateo del gráfico.



La entrada de datos a la aplicación Microsoft Chart puede realizarse manualmente, o bien trasladando bloques de datos elaborados por otras aplicaciones como, por ejemplo, la hoja electrónica Multiplan.



Una vez introducidos los datos a representar, el usuario decide qué tipo de diagrama desea obtener entre las múltiples opciones que brinda la aplicación.

Dependiendo de la opción seleccionada por el usuario, el programa mostrará un nuevo menú para elegir entre las distintas posibilidades asociadas a cada caso.

ELEMENTOS GRAFICOS DEL MICROSOFT CHART

El resultado final de la ejecución del programa, estará compuesto por cuatro elementos autónomos: los ejes del diagrama, las etiquetas de los ejes, las leyendas y el gráfico propiamente dicho. Todos ellos son controlados interactivamente por el usuario con las siguientes posibilidades:

- *Ejes del diagrama*

Todos los diagramas, excepto las tartas, se representan sobre un sistema de dos ejes coordenados. Las categorías se situarán sobre el eje horizontal (abscisas) y los valores sobre el eje vertical (ordenadas). La escala de los ejes la calcula automáticamente el programa, de forma que aparezcan correctamente todos los valores.

No obstante, el usuario puede modificar cualquiera de los atributos de los ejes utilizando los comandos oportunos; por ejemplo, el operador puede hacer que un diagrama de líneas se ajuste a una escala logarítmica, puede cambiar el intervalo de separación entre las categorías, etc. Todas las opciones para gestionar los ejes se pueden invocar desde el menú FORMAT.

Otra característica importante del MICROSOFT CHART, que da una idea de su adaptabilidad a cualquier situación, es que permite al usuario decidir en qué punto se deben cortar los dos ejes.

- *Etiquetas de los ejes*

Una vez decididas las características generales de los ejes, estos se marcarán con etiquetas según la escala. El propio usuario puede decidir la distancia de separación entre ellas, su ausencia, o cualquier otra característica apelando a las opciones del menú FORMAT.

Cabe elegir, incluso, la posibilidad de que todo el gráfico aparezca sobre una red de líneas o cuadrícula, cuya presencia facilitará la localización de categorías y valores a partir de un punto concreto del gráfico.

En cuanto a los formatos con los que pueden aparecer escritas las marcas relativas a los ejes, cabe destacar los siguientes:

— Formato general (p.e. 48.7)
— Formato dollar (p.e. $48.70)
— Formato porcentual (p.e. 4870.00%)
— Formato entero (p.e. 49)
— Formato decimal (p.e. 48.70)
— Formato literal (p.e. "ENERO")

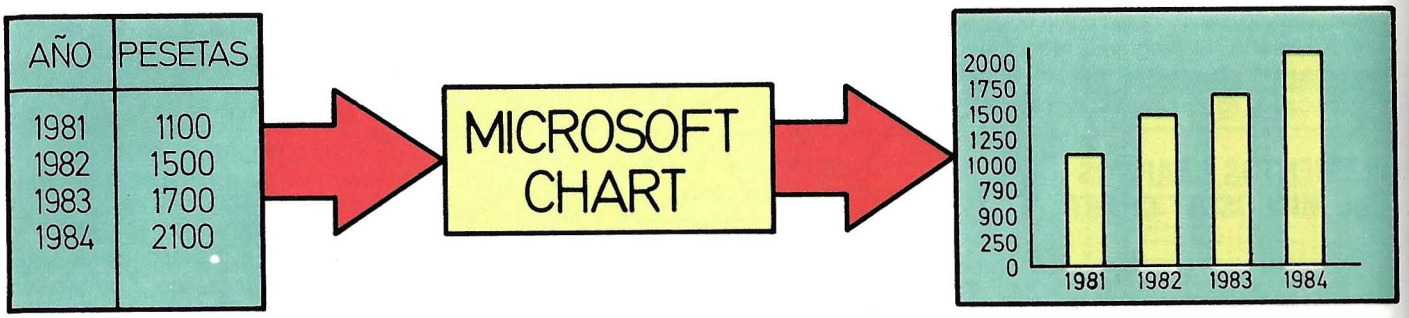
En el caso de que el usuario desee utilizar algún formato particular, puede crearlo a voluntad. Por ejemplo, se podría definir un formato que añadiera a continuación de cada número la abreviatura "Pts".

Si se trabaja con fechas, el programa está capacitado para manejar las siguientes unidades: AÑO, TRIMESTRE, MES, DIA y DIA/SEMANA. De esta forma, si se desea, por ejemplo, trabajar en trimestres, comenzando por el 3.er TRIM-84, y se decide un incremento unitario, el programa será capaz de generar la serie: 3.er TRIM-84, 4.º TRIM-84, 1.er TRIM-85, 2.º TRIM-85...

- *Leyendas del diagrama*

Mediante los comandos apropiados del menú CHART, el usuario puede añadir textos dentro del gráfico. Normalmente, las leyendas se utilizan para que en el diagrama aparezca escrito el nombre de las categorías sobre el eje horizontal y el de los valores en el eje vertical. Sin embargo, también puede utilizarse esta facultad para crear cualquier otro tipo de literal. Por ejemplo: para el título general del gráfico (que se puede situar en la parte superior o en la inferior del dibujo), para incluir comentarios que resalten las propiedades de los fenómenos estudiados...

Microsoft Chart es un paquete de aplicación estándar, de la firma americana Microsoft, orientado a la puesta a punto de gráficos de gestión.



En términos generales, la misión de un paquete para gráficos de gestión —como es el caso del Microsoft Chart—, no es otra que convertir en diagramas los datos aportados por el usuario.

La facilidad de manejo de los comandos para la creación de leyendas es muy grande. En la práctica, el operador decide en qué posición debe comenzar el literal; teclea los caracteres que lo forman y, automáticamente, aparece la leyenda en el diagrama. Si tras observarla en la pantalla no resulta del agrado del usuario, éste puede modificarla, cambiarla de posición o borrarla.

- *Gráfico*

La parte más importante del diagrama es, sin lugar a dudas, el gráfico que se basará en los datos introducidos por el usuario. Para su captación, el programa presentará una «ventana» con dos líneas verticales: una para las categorías y otra para los valores; a través del teclado se introducirán el número o literal que corresponda en la línea en la que esté situado el cursor. La posterior corrección de los datos es auxiliada por el editor que aportará facilidades para la gestión de los datos, permitiendo insertar valores, eliminarlos...

Una vez conformes con los datos introducidos, queda por elegir el tipo de representación con la que se desea que aparezcan en la pantalla, cosa que puede realizarse desde el menú GALLERY. Al seleccionarlo, éste presentará en pantalla una relación de los diagramas disponibles (superficies, barras, tartas, líneas). Una vez realizada la elección por parte del operador, aparecerán una serie de ejemplos sobre las distintas posibilidades admisibles en el diagrama relacionado; posibilidades entre las que el usuario puede elegir la más apropiada según el caso. Cabe también la alternativa de combinar varios diagramas en un único gráfico.

Esta posibilidad resulta muy útil para realizar composiciones que resalten distintas peculiaridades de los fenómenos integrados en una única imagen.

ALMACENAMIENTO E IMPRESION DE LOS GRAFICOS

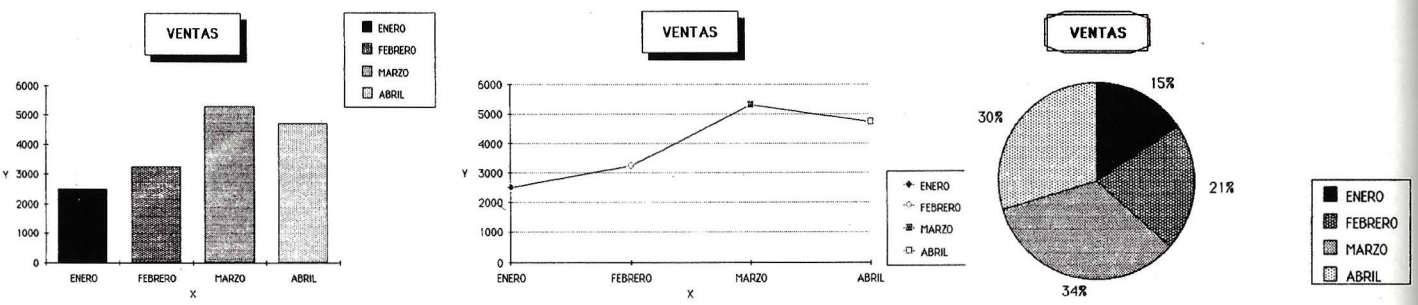
Habitualmente, después de haber producido el gráfico en la pantalla del ordenador se desea grabar la información en un disco flexible o en una unidad de disco rígido; o tal vez reproducir el gráfico sin mayor dilación mediante la impresora. Todas estas son opciones contempladas en el menú FILE, cuya misión general es la comunicación del programa con los periféricos de almacenamiento y de impresión.

La impresión de los gráficos puede obtenerse sobre papel continuo o en hojas separadas; en cualquier caso, el programa permite al usuario determinar ciertas características relativas al tamaño, calidad y otros atributos del diagrama.

Por lo que respecta a la posibilidad de almacenamiento, la información que se grabará sobre el soporte magnético serán los valores numéricos y alfabéticos que caracterizan al gráfico en su totalidad y no el gráfico propiamente dicho, no obstante existen versiones que permiten almacenar la información con formatos externos al programa.

Se puede afirmar que en la mayoría de las aplicaciones del MICROSOFT CHART, y de cualquier paquete gráfico en general, la misión fundamental consiste, simplemente, en la representación de los gráficos en la pantalla y sólo en casos concretos resulta necesario producir su copia en papel o en soportes de almacenamiento.

En el próximo capítulo proseguirá el estudio del paquete de aplicación MICROSOFT CHART, haciendo especial hincapié en los aspectos prácticos.



Un mismo grupo de valores puede representarse gráficamente de muy diversas formas, a voluntad del usuario. Los tres diagramas ilustrados corresponden a la representación de un mismo conjunto de datos. En este caso se reproducen las copias en papel obtenidas por medio de la impresora asociada a un Apple Macintosh; por supuesto, equipado con la aplicación Microsoft Chart.

La máquina divertida

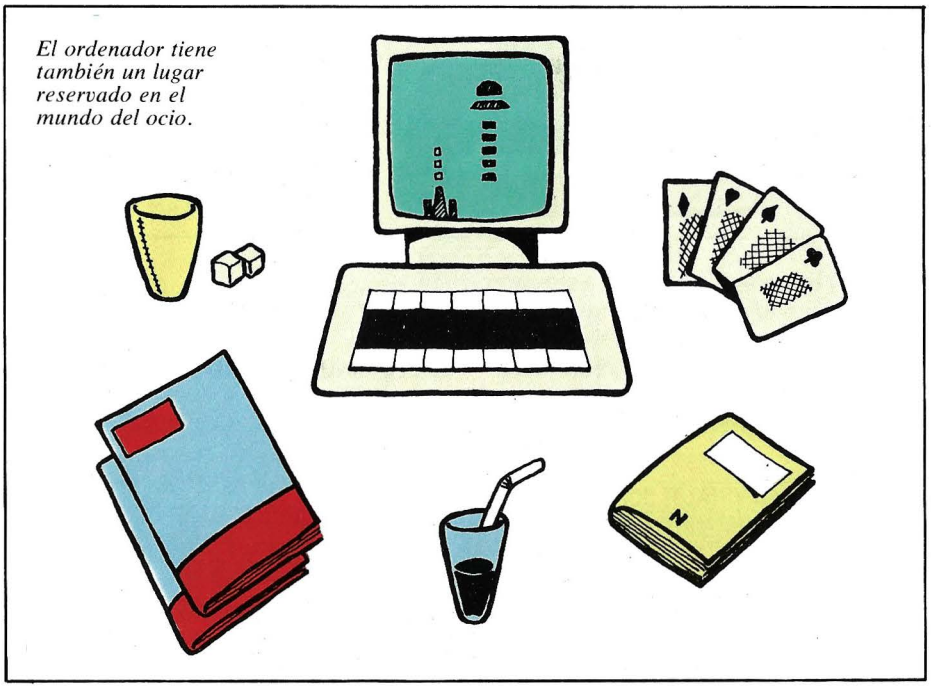
Jugando con el ordenador

El ordenador es un aparato capaz de realizar múltiples funciones. Con él se pueden solucionar cálculos complejos y manejar grandes volúmenes de información. Pero también es posible jugar con un ordenador.

El aspecto lúdico es una de las facetas más explotadas en los ordenadores domésticos. Existe una gran cantidad de juegos comercializados para los distintos aparatos. Juegos no sólo sirven para llenar los ratos de ocio del usuario; ahí están, por ejemplo, los juegos educativos. Con éstos, los más jóvenes, e incluso los ya adultos, ejercitan su mente de una forma amena y divertida.

La amplia proliferación de juegos para ordenador hace de éstos un moderno objeto de intercambio cultural. Sin duda, a ellos se debe en gran medida el éxito de los ordenadores domésticos.

El BASIC es un lenguaje con el que el diseño y realización de juegos resulta relativamente fácil, tal y como se observará a lo largo el presente capítulo.



El ordenador tiene también un lugar reservado en el mundo del ocio.

UN JUEGO SENCILLO

Con las herramientas que brinda el BASIC es tarea fácil construir un programa que amenice los ratos muertos del operador, por ejemplo el siguiente. El ordenador "piensa" un número, que debe ser adivinado por el jugador. Para dar una pista el ordenador informa si el número secreto es mayor o menor que el propuesto.

La codificación en BASIC de este programa resulta de lo más simple. En primer

El microordenador puede convertirse en un compañero de juego ideal. Sólo hace falta programarlo correctamente.

En el primer programa ejemplo, el ordenador "reta" al usuario para que adivine el número secreto elegido al azar.

lugar, hay que elegir el número. Este debe estar comprendido entre ciertos límites, por ejemplo 0 y 100. Para ello se hará uso de la capacidad de generación de números aleatorios del aparato, plasmada en la función RND:

```
20 LET MINUM=INT(100*RND)
```

```
10 REM ADIVINALO! SOFT
20 LET MINUM=INT(100*RND)
30 PRINT"HE PENSADO UN NUMERO DEL
   CERO AL CIEN"
100 INPUT "CUAL ES";TUNUM
110 IF TUNUM>MINUM THEN
    PRINT"NOOO!! ES MENOR"
120 IF TUNUM<MINUM THEN
    PRINT"NOOO!! ES MAYOR"
130 IF TUNUM=MINUM THEN
    GOTO 200
140 GOTO 100
200 PRINT"MUY BIEN,ERA EL ";MINUM
210 PRINT"JUGAMOS OTRA VEZ (S/N)?"
220 LET A$=INKEY$
230 IF A$="S" THEN GOTO 20
240 IF A$<>"N" THEN GOTO 220
250 PRINT"HASTA LA PROXIMA!"
```

Con esta única línea se obtiene dicho número, ya que al multiplicar RND por 100 se obtiene un valor mayor o igual que 0 y menor que 100. De éste se toma la parte entera para descartar todos los posibles valores que tuvieran parte fraccionaria. Una vez que se dispone del número a desvelar hay que brindar al usuario la posibilidad de acertarlo.
Como se tratará (muy posiblemente) de un número de más de un dígito no conviene utilizar INKEY$, sino que es preciso optar por el comando INPUT.

```
100 INPUT "CUAL ES";TUNUM
```

Ahora hay que comparar el valor supuesto con el verdadero, e indicar si el número secreto es mayor o menor. En caso contrario es que ambos números serán coincidentes y, en consecuencia, será preciso felicitar al jugador.

```
110 IF TUNUM>MINUM THEN
    PRINT "NOOO!! ES MENOR"
120 IF TUNUM<MINUN THEN
    PRINT "NOOO!! ES MAYOR"
130 IF TUNUM=MINUM THEN
    PRINT "MUY BIEN,ERA EL ";MINUM
```

Con estas líneas se consiguen distintos mensajes dependiendo del número aportado. Si no se acierta al menos se obtiene una pista. Aunque de poco sirve esa pista si no se pueden seguir probando números. Si el intento es infructuoso, habrá que repetirlo. Es evidente pues que hace falta un GOTO 100. Para situarlo debidamente es necesario hacer una pequeña reestructuración:

```
130 IF TUNUM=MINUM THEN GOTO 200
140 GOTO 100
```

Ahora, el número correcto desviará la ejecución hacia la línea 200. Si el número es mayor o menor (distinto) del pensado por la máquina, se efectuará el salto a la línea 100.
A partir de la línea 200 hay que escribir la felicitación y dar por concluido el juego.

```
200 PRINT"MUY BIEN,ERA EL ";MINUM
```

Aunque un detalle de buen gusto consiste en permitir al usuario que repita el juego (con otro número, por supuesto). Al efecto se introducirán las líneas oportunas que permitan al operador elegir entre vol-

ver a jugar o abandonar el programa. Estas son:

```
210 PRINT"JUGAMOS OTRA VEZ (S/N)?"
220 LET A$=INKEY$
230 IF A$="S" THEN GOTO 20
240 IF A$<>"N" THEN GOTO 220
250 PRINT"HASTA LA PROXIMA!"
```

Las líneas 210 y 220 se encargan de recoger la petición del jugador. Si éste responde con "S" (Sí) se reinicia el programa en la línea 20. De no ser ésta su respuesta, el programa salta a la línea 240. En ella se bifurca a la línea 220 si no contestó con una "N". Por lo tanto, si no se responde con "S" o "N", se recogerá un nuevo carácter del teclado. Cuando la respuesta es una pulsación de la tecla N, se continúa en la línea 250. En tal caso, el ordenador se despide dando por concluida la ejecución del programa.
Por último, sería interesante indicar al principio del juego el rango de números a considerar como posibles. Para ello basta con incluir la siguiente línea:

```
30 PRINT"HE PENSADO UN NUMERO DEL
   CERO AL CIEN"
```

El programa completo es el que aparece en el cuadro adjunto bajo el título "ADIVINALO!".
Una posible partida seguirá la evolución siguiente:

```
RUN
HE PENSADO UN NUMERO DEL CERO
  AL CIEN
CUAL ES? 52<CR>
NOOO!! ES MENOR
CUAL ES? 23<CR>
NOOO!! ES MAYOR
CUAL ES? 45<CR>
MUY BIEN,ERA EL 45
JUGAMOS OTRA VEZ (S/N)?
HASTA LA PROXIMA!
```

UNA PARTIDA A LOS CHINOS

Adoptando algunas de las ideas plasmadas en el ejemplo anterior, es posible redactar un nuevo programa con el que el usuario se puede jugar el aperitivo. Se trata, esta vez, de enseñar al ordenador a jugar al popular deporte de "los chinos". Ello se reduce a adivinar un número del 0 al 3 y sumarlo al previamente pensado. El problema se puede desglosar en varias partes:

— Elegir un número (del 0 al 3)
— Pedir vaticinio del jugador (del 0 al 6)
— Calcular el rango de valores posibles
— Elegir un número dentro de ese rango.

Los dos primeros pasos son similares a los del ejemplo anterior. La única variación está en los límites del valor elegido:

```
20 LET MIAS=INT(4*RND)
30 INPUT "CUANTAS" ;T
```

El margen de posibles valores será, en principio, desde MIAS hasta MIAS+3. Para elegir un valor aleatorio dentro de estos límites basta con la siguiente línea:

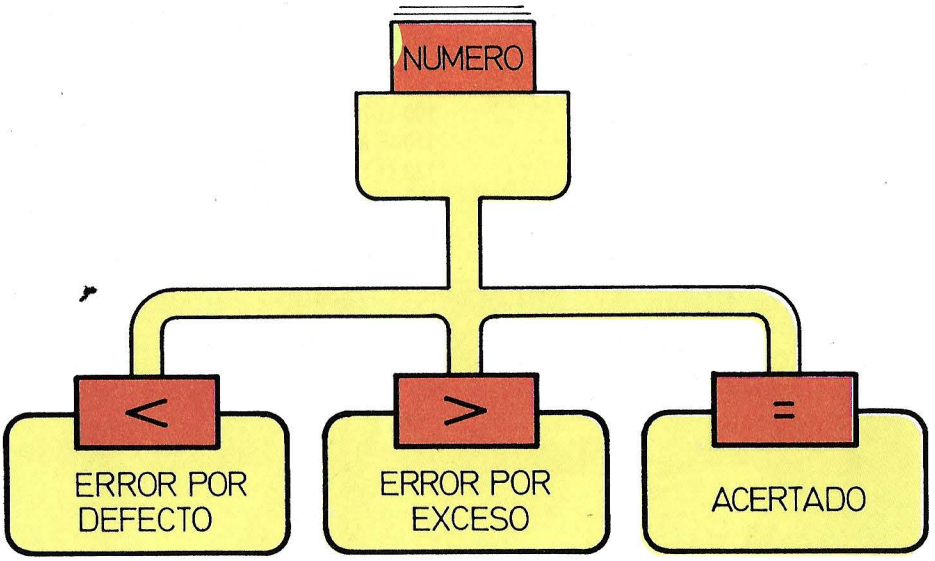
```
50 LET Y=MIAS+INT(4*RND)
```

Este cálculo debe repetirse hasta encontrar un número distinto del proporcionado por el adversario, lo que se consigue introduciendo un comando IF:

```
60 IF Y=T THEN GOTO 50
```

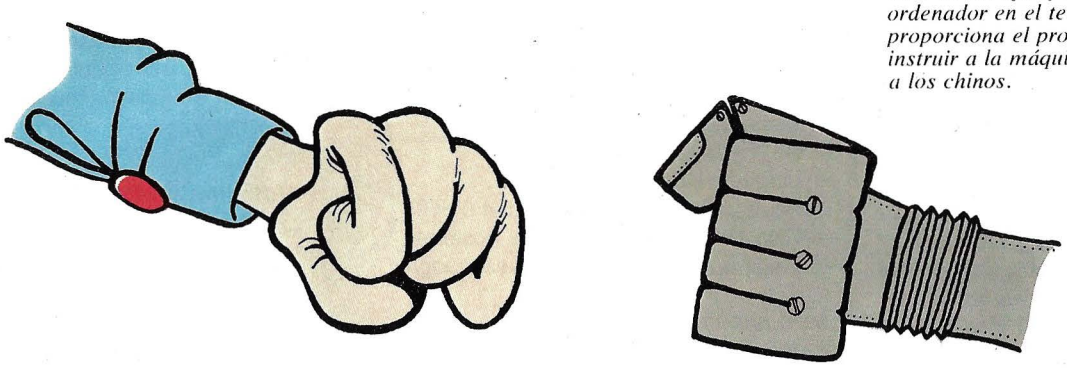
Si el número Y elegido coincide con T (el pronóstico del oponente) se vuelve a elegir un nuevo Y. Sólo se continuará en el caso de que ambos datos difieran.
El programa completo necesita un par de líneas más para hacer público su pronóstico y mostrar su jugada (líneas 70 y 80):

```
10 REM CHINOS 1/SOFT
20 LET MIAS=INT(4*RND)
30 INPUT "CUANTAS";T
50 LET Y=MIAS+INT(4*RND)
60 IF Y=T THEN GOTO 50
70 PRINT "DIGO QUE HAY ";Y
80 PRINT "LLEVO ";MIAS
■
```



Esquema lógico del programa "¡ADIVINALO!" que constituye el primero de los juegos programados en el texto.

Desde luego, se puede sacar alguna información adicional del dato proporcionado por el contrincante. Si pide 0 es que no lleva ninguna, si pide 1 es que lleva 0 ó 1, etc. Esto se puede abreviar diciendo que si el contrario pide N es que sólo puede llevar un número comprendido entre 0 y N. Axioma éste sólo válido para N menor que 4.
Si el contrario estima que el número correcto es 4 es porque él lleva al menos una. Con esta lógica se observa que, dependiendo del dato adquirido, se puede limitar el rango de valores.



Otro de los ejemplos de la eficacia del ordenador en el terreno del ocio, lo proporciona el programa capaz de instruir a la máquina para que juegue a los chinos.

Así pues, los límites de los posibles valores serán:

— *Mínimo:* N−3 ó 0
— *Máximo:* N ó 3

DATO	POSIBLES
0	0
1	01
2	012
3	0123
4	123
5	23
6	3

Lo que convertido en líneas de programa, se traduce en:

```
100 LET MIN=T−3
110 IF MIN<0 THEN MIN=0
120 LET MAX=T
130 IF MAX>3 THEN MAX=3
```

El lenguaje BASIC, presente en la mayor parte de los ordenadores personales, constituye también una potente herramienta para la programación de juegos.

Las líneas 100 y 120 calculan los valores máximo y mínimo según las fórmulas comentadas más arriba. Las otras dos líneas, 110 y 130, impiden que dichos valores se sitúen por debajo de 0 (el mínimo) o por encima de 3 (el máximo).
Una vez obtenido el rango apropiado, es preciso elegir uno de los números admisibles. Al respecto se utilizará la fórmula comentada en el capítulo dedicado a la función RND:

```
INT(RND*(MAX−MIN+1))+MIN
```

En ella, todos los valores son conocidos. Lo que interesa es adivinar el número total de monedas, por lo que, al número hallado, ha de sumársele aquel que juega la máquina. Todo ello se almacena en la variable Y para su posterior manipulación.

```
200 LET Y=MIAS+INT(RND*(MAX
    −MIN+1))+MIN
```

Claro que el pronóstico así evaluado puede coincidir con el del contrincante. Si eso ocurre habrá que buscar otro valor, recurriendo al mismo método utilizado en el programa anterior:

```
210 IF Y=T THEN GOTO 200
```

Esta simple línea hace que se repita el cálculo hasta que el número resultante sea distinto del proporcionado por el jugador. No obstante, esta actuación plantea un inconveniente: si el rango de valores posibles se limita a un solo número, y éste coincide con el elegido por el adversario, la ejecución entrará en un bucle cerrado.

En efecto, el ordenador calculará siempre el mismo valor Y que coincide con T, y saltará de nuevo a la línea 200.
Este tipo de bucles que no tienen salida (al igual que los del tipo 100 GOTO 100) se denominan "bucles infinitos". El programador debe huir de estos bucles, ya que introducen al programa en un "callejón sin salida".
Así pues, sólo debe recalcularse el pronóstico cuando el rango incluya al menos 2 valores posibles. O lo que es lo mismo, cuando MIN y MAX tomen valores distintos. La anterior línea 200 se sustituiría, en consecuencia, por la que sigue:

```
210 IF Y=T AND MIN<>MAX THEN GOTO 200
```

Ahora la ejecución puede continuar de dos posibles formas:
— Al calcular un número distinto del ofrecido por el oponente.
— Cuando el contrario ha elegido el único valor posible.
En el segundo caso no cabe duda de que el jugador humano ha acertado. Dicha situación se dará cuando, al pasar a la siguiente línea, los valores de T e Y sean idénticos. En ese momento, al ordenador no le queda más remedio que rendirse ante la evidencia de su derrota.

```
220 IF Y=T THEN PRINT "ACERTASTE"
```

La otra posibilidad es que seleccione un valor distinto al del contrincante (T e Y son distintos). Ello indica que todo va bien y el ordenador puede emitir su pronóstico y mostrar su jugada:

```
230 IF Y<>T THEN PRINT "YO DIGO ";Y
300 PRINT "LLEVO ";MIAS
```

Si el usuario quiere ahorrarse el trabajo de contar las fichas de uno y otro participante e ir llevando el tanteo acumulado, ha de completar el programa con algunas líneas más. En primer lugar, el ordenador debe conocer el número de fichas jugado por el oponente, para hallar la suma total. La recogida de este dato se realizará por medio de un comando INPUT.

```
290 INPUT "CUANTAS LLEVAS";TUYAS
```

Como se observa, esta línea se ha colocado inmediatamente antes de que el aparato muestre su jugada (línea 300). De esta forma se evitan cambios de última hora por parte del operador humano.
Una vez conocido este dato, hay que proceder a su suma y posterior comparación con los pronósticos de ambos jugadores.

```
310 LET S=TUYAS+MIAS
320 IF S=Y THEN PRINT "GANO YO"
330 IF S=T THEN PRINT "GANAS TU"
```

En las líneas 320 y 330 es donde se compara el resultado final con los números propuestos por los jugadores. Si alguno de ellos acierta, se informa quién es el ganador.
También es posible llevar la cuenta de los aciertos de cada contendiente. Para ello basta con incrementar un contador particular cada vez que tenga lugar ese hecho. El lugar apropiado para introducir las órdenes pertinentes es el argumento de THEN

Esquema de tareas a realizar para construir el programa del "juego de los chinos".

```
5 REM CHINOS/SOFT
10 LET HU=0:LET MI=0
20 LET MIAS=INT(4*RND)
30 INPUT "CUANTAS";T
100 LET MIN=T-3
110 IF MIN<0 THEN MIN=0
120 LET MAX=T
130 IF MAX>3 THEN MAX=3
200 LET Y=MIAS+INT(RND*(MAX-MIN
    +1))+MIN
210 IF Y=T AND MIN<>MAX THEN
    GOTO 200
220 IF Y=T THEN PRINT "ACERTASTE"
230 IF Y<>T THEN PRINT "YO DIGO ";Y
290 INPUT "CUANTAS LLEVAS";TUYAS
300 PRINT "LEVO";MIAS
310 LET S=TUYAS+MIAS
320 IF S=Y THEN PRINT "GANO YO":LET
    MI=MI+1
330 IF S=T THEN PRINT "GANAS TU":LET
    HU=HU+1
400 PRINT "HUMANO:";HU;"MICRO:";MI
1000 GOTO 20
```

de las líneas 320 y 330. Para poder incluir en ellos tanto la impresión del mensaje como la actualización del contador, se hará uso de la posibilidad de encadenar varias órdenes en la misma línea. Esto, como ya se vio, se realiza separándolas con el símbolo dos puntos (:).
Así pues, en las dos nuevas líneas las variables MI y HU llevan la cuenta de las partidas ganadas por el "micro" y el "humano", respectivamente:

```
320 IF S=Y THEN PRINT "GANO YO":LET
    MI=MI+1
330 IF S=T THEN PRINT "GANAS TU":LET
    HU=HU+1
400 PRINT "HUMANO:";HU;"MICRO:";MI
```

Se ha incluido la línea 400 que muestra la puntuación alcanzada por los jugadores hasta ese momento. Para evitar que ninguno de los participantes empiece con ventaja, es necesario inicializar a cero ambos contadores al principio de la partida:

```
10 LET HU=0:LET MI=0
```

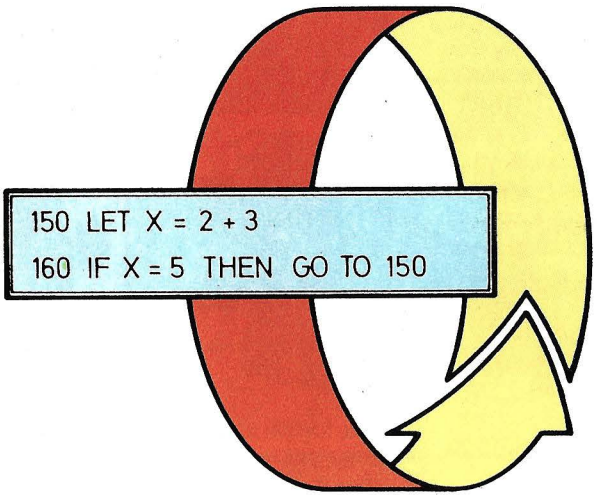
Por último, al final del programa hay que incluir una instrucción que permita volver a jugar. El salto se realiza a la línea 20, en la que empezará una nueva partida.

```
1000 GOTO 20
```

Queda al gusto del lector la inclusión de una rutina de espera entre las líneas 400 y 1000; rutina que otorgue el tiempo suficiente para elegir la siguiente jugada. El programa completo (CHINOS) aparece listado en el cuadro adjunto.

UN JUEGO DE HABILIDAD

El siguiente programa consiste en un juego de habilidad. El jugador ha de disparar a un blanco móvil. Dicho blanco es un transporte enemigo que debe ser destruido a toda costa.
El problema se va a afrontar con un criterio de modularidad, diseñando cada parte por separado, para al final unirlas todas.
El vehículo se desplazará a través del borde superior de la pantalla. Ello significa



Los bucles "infinitos" son estructuras cíclicas que encierran al flujo del programa, impidiendo que evolucione la secuencia de proceso. El programador debe evitar su formación accidental.

La proliferación de los ordenadores domésticos se ha visto apoyada en gran medida por su idoneidad para las aplicaciones lúdicas.

que habrá que dibujar el blanco sucesivamente en las distintas posiciones. Las líneas que dibujan el vehículo son las siguientes:

```
110 PRINT AT(X,Y) "LLL"
120 PRINT AT(X,Y) "0 0"
```

Los parámetros Y de la función AT adoptarán los valores 1 y 2, respectivamente, para que el blanco se mantenga en las dos líneas superiores. El parámetro X variará desde 1 hasta el valor que lleve al vehículo al extremo de la pantalla. La variación se consigue encerrando ambas instrucciones dentro de un bucle FOR/NEXT.

```
100 FOR X=1 TO 25
110 PRINT AT(X,1) "LLL"
120 PRINT AT(X,2) "0 0"
130 NEXT X
```

Esta zona del programa hará que se desplace el blanco. Pero, al no borrarse la posición anterior persistirá en la pantalla el rastro del recorrido:

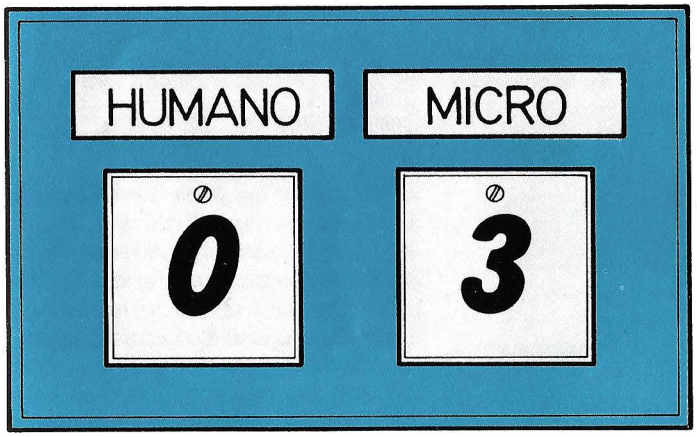
```
RUN
LLLLLLLLL LLL
0000000000 0 0
■
```

Esto se soluciona introduciendo, sencillamente, un espacio en blanco al principio de cada una de las cadenas que representan al vehículo.

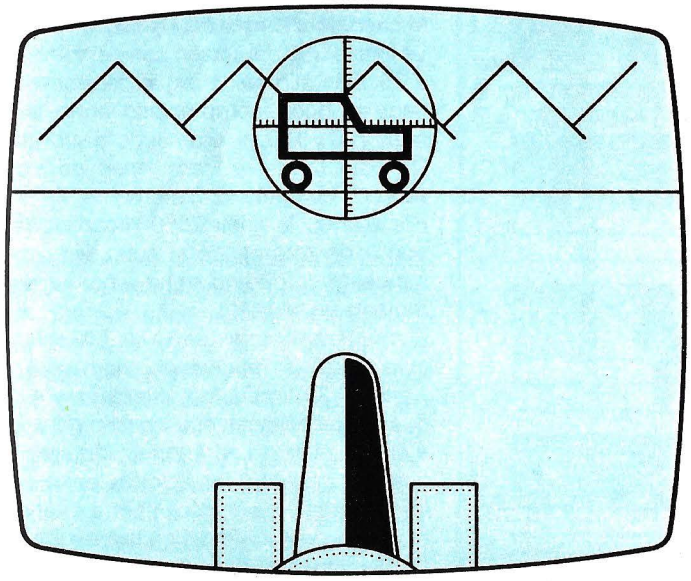
```
100 FOR X=1 TO 25
110 PRINT AT(X,1) " LLL"
120 PRINT AT(X,2) " 0 0"
130 NEXT X
```

De esta forma, los espacios borrarán la imagen de las posiciones precedentes.

```
RUN
 LLL
 0 0
■
```



Y no sólo es capaz de suplir al más avezado contendiente en los juegos. Además de jugar, el ordenador también puede controlar la evolución de las partidas y actualizar el marcador.



Escenario de la batalla programada como tercer ejemplo de juego con el ordenador.

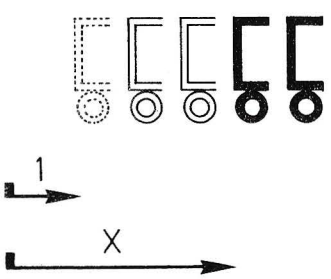
Con el enemigo ya funcionando, lo que hay que hacer ahora es dibujar nuestro cañón. Se tomará como representación del mismo un carácter cuyo aspecto recuerde la forma del mismo. La letra I puede servir para ese propósito. Esta se colocará en un punto central de la zona inferior de la pantalla. Por ejemplo, en la columna 15 de la vigésima fila. Dichos valores, al igual que el 25 del bucle FOR anterior, pueden ser modificados al gusto del usuario. La línea que sitúa el cañón puede ser la siguiente:

```
50 PRINT AT(17,20)"I"
```

Ahora viene la rutina de disparo. Esta se utilizará a modo de subrutina, por lo que es conveniente separarla del resto del programa.

```
1000 FOR I=19 TO 2 STEP -1
1005 FOR J=0 TO 2
1010 PRINT AT(17,Z)"!"
1020 PRINT AT(17,Z)" "
1025 NEXT J
1030 NEXT I
```

Esta rutina traza la trayectoria del proyectil empleando el carácter !. Su desplazamiento tiene lugar a lo largo de la columna 17 de la pantalla, columna que corres-




```
FOR X = 1 TO 25
PRINT AT (X,2) "<Vehiculo>"
NEXT X
```

El movimiento del vehículo enemigo se simula emplazando a éste, sucesivamente, a mayor distancia del borde izquierdo de la pantalla.

ponde a la posición del cañón. El bucle en J hará que el proyectil parpadee en cada posición antes de desaparecer.
La rutina de disparo únicamente se ha de ejecutar cuando se pulse una tecla. Para ello, es preciso incluir algunas líneas que reconozcan la pulsación. Líneas que deben estar localizadas en la zona del programa que mueve el blanco; en caso contrario, el disparo sólo podría efectuarse cuando el enemigo ya ha rebasado la posición del cañón.

```
122 LET T$=INKEY$
124 IF T$<> THEN GOSUB 1000
```

La rutina de disparo no está completa aún. Es necesario detectar si se ha producido una colisión entre el proyectil y el blanco. Tal operación se realiza al final de la rutina de disparo. Lo único que hay que hacer es cerciorarse de que el objetivo se encuentra (o no se encuentra) en la vertical del cañón. Este hecho se produce cuando la variable X adquiere los valores 16, 15 ó 14.

```
10 REM BATALLA/SOFT
20 LET PUNTOS=0
30 CLS
40 PRINT AT(1,20)"PUNTOS ";PUNTOS
50 PRINT AT(17,20)"I"
100 FOR X=1 TO 25
110 PRINT AT(X,1) "LLL"
120 PRINT AT(X,2) "0 0"
122 LET T$=INKEY$
124 IF T$<>" "THEN GOSUB 1000
130 NEXT X
150 GOTO 30
1000 FOR I=19 TO 3 STEP -1
1005 FOR J=0 TO 2
1010 PRINT AT(17,I)"!"
1020 PRINT AT(17,I) " "
1025 NEXT J
1030 NEXT I
1100 IF X>13 AND X<17 THEN GOTO 2000
1110 RETURN
2000 FOR J=0 TO 50
2010 PRINT AT(17,1)"*"
2020 PRINT AT(17,1)" "
2030 NEXT J
2040 LET PUNTOS=PUNTOS+1
2050 X=25
2060 RETURN
```

Así pues, se plantea una condición que de cumplirse indicará que se ha alcanzado el objetivo:

```
1100 IF X>13 AND X<17 THEN GOTO 2000
1110 RETURN
```

El efecto de ambas líneas desvía la ejecución hacia la línea 2000 si se produce la colisión, u ordena el retorno al programa principal en el caso contrario. Su actuación permite el tratamiento aparte (línea 2000) de la colisión. Dicho tratamiento consistirá en visualizar la explosión y reiniciar el programa; desde luego, también se puede actualizar un contador que indique el número de blancos acertados.

```
2000 FOR J=0 TO 50
2010 PRINT AT(17,1)"*"
2020 PRINT AT(17,1)" "
2030 NEXT J
2040 LET PUNTOS=PUNTOS+1
2050 X=25
2060 RETURN
```

En estas últimas líneas se ha incluido un bucle (líneas de la 2000 a la 2030) que imprime y borra, alternativamente, un asterisco en el punto de impacto del proyectil. Ello produce un efecto de explosión. El bucle se repite el número de veces suficiente como para que se aprecien los daños producidos.
La línea 2040 es la encargada de incrementar en una unidad el controlador de aciertos. Como se puede observar, la actualización de PUNTOS sólo se efectúa en el caso de alcanzar el objetivo.
La línea 2050 del listado pone el valor de X a 25. Con ello se consigue dar por terminado el bucle comprendido entre las líneas 100 y 130, al retornar de la subrutina. De esta forma, el carro enemigo alcanzado no continúa su marcha.
Por último, la línea 2060 proporciona el punto de retorno de la subrutina, en el supuesto de que no se haya producido el regreso en la 1110.
El programa completo (BATALLA) exhibirá el aspecto que muestra el cuadro adjunto.
La línea 30 limpia la pantalla para evitar que se superponga el escenario del juego a cualquier texto anterior. La siguiente línea (40) muestra los aciertos en la zona inferior izquierda de la pantalla. La variable PUNTOS es inicializada en la línea 20. Por fin, la línea 150 reinicia la ejecución cuando no se destruye el objetivo.

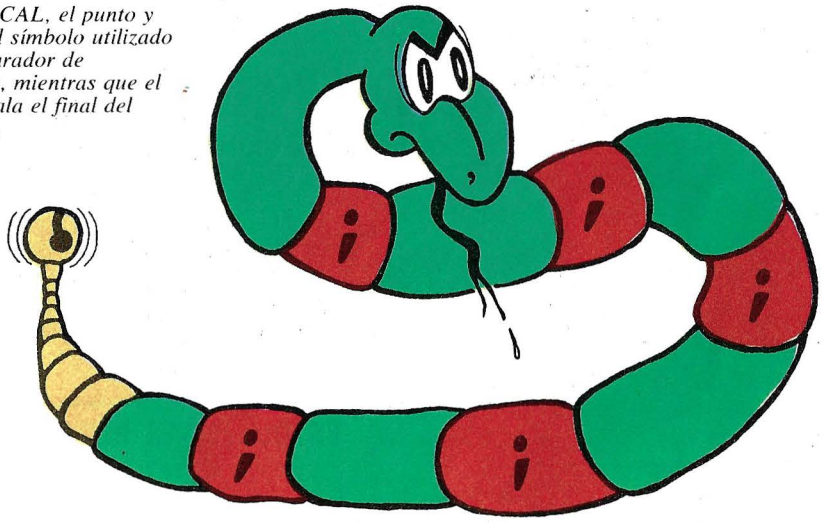
Pascal (4)

Estructuras y sentencias de control

La programación estructurada aporta un conjunto de técnicas que permiten una fácil depuración y puesta a punto de los programas. Estos resultan inteligibles para otras personas distintas de programador y susceptibles de recibir posteriores modificaciones.

Al igual que ocurre con todos los lenguajes de programación, un programa en PASCAL se puede codificar por medio de una sucesión de sentencias que se van ejecutando ordenadamente, una tras otra, siguiendo las directrices de lo que se llama *flujo del programa*. La continuidad del flujo del programa está garantizada en el PASCAL por su carácter de lenguaje estructurado. Ello hace que sea posible preservar la secuencia de ejecución y, por lo tanto, que no se distraiga la atención intentando comprender la lógica del programa siguiendo los saltos incondicionales de una a otra parte del mismo.

En el PASCAL, el punto y coma es el símbolo utilizado como separador de sentencias, mientras que el punto señala el final del programa.



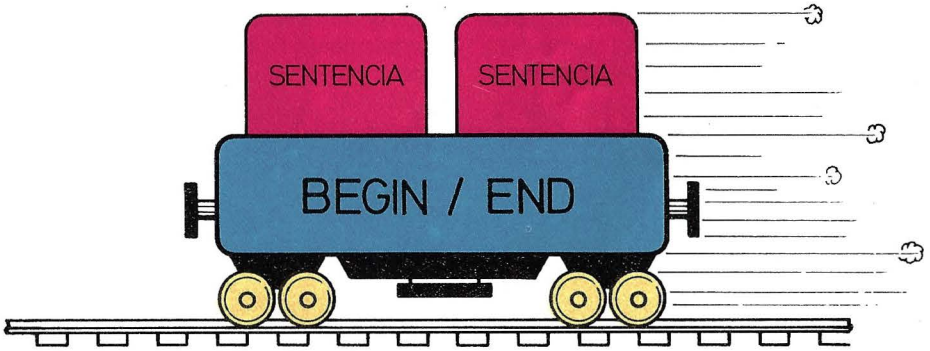
ESTRUCTURA SECUENCIAL

Recordando los conceptos de la programación estructurada, la primera estructura que nos encontramos es la *secuencial*. En el PASCAL, una estructura secuencial está formada por sentencias de acción, que no desvían el flujo del programa; tal es el caso de las sentencias de asignación, de entrada y salida de datos... El programa en su conjunto se puede considerar como una sentencia única, al estar englobado el cuerpo del programa entre las palabras reservadas BEGIN y END.

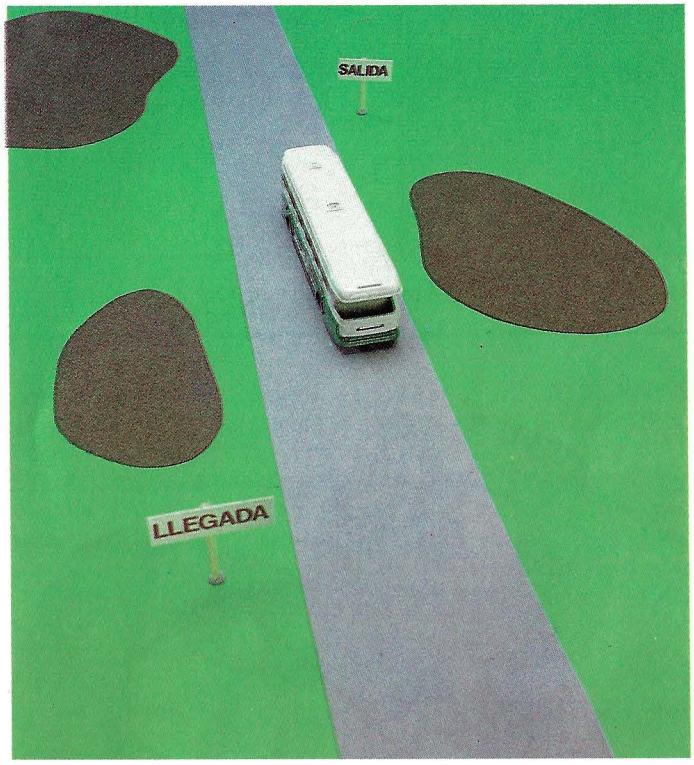
El bloque BEGIN/END conforma una sentencia compuesta, de manera que las sentencias encerradas en su interior se consideran como una sola. Esta es una estructura muy utilizada, como se verá más adelante.

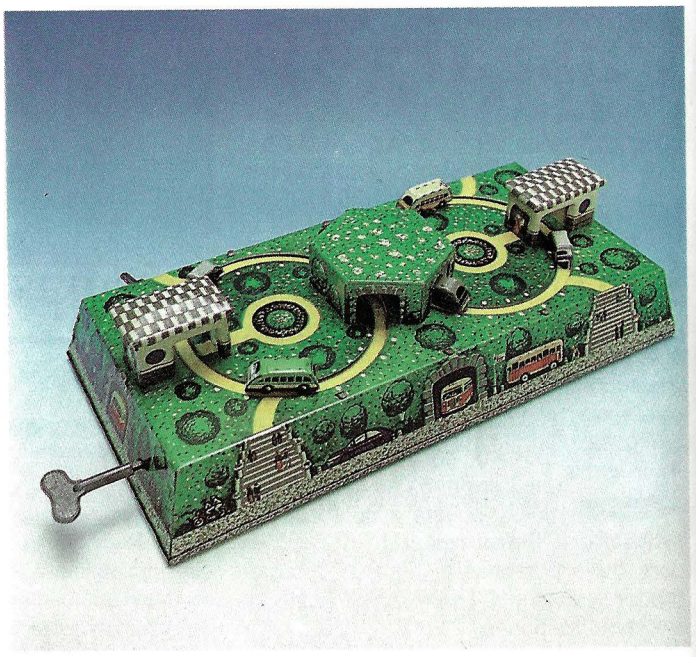
Algunas estructuras de control necesitan llevar asociada una única sentencia y, sin embargo, la propia lógica del programa exige la presencia de varias de ellas para que sea posible programar la tarea implicada; en tal caso, al incluir a todas las sentencias necesarias en un único bloque BEGIN/END, la estructura de control "creerá" que sólo tiene una sentencia asociada.

El uso de sentencias compuestas dentro de un programa debe limitarse exclusivamente a los casos en los que sea plenamente necesario. Aunque el abuso de ellas no produzca ningún tipo de error, es



Una sentencia compuesta, delimitada por las palabras clave BEGIN y END, puede englobar a múltiples sentencias elementales. A pesar de ello, la máquina trata a las sentencias compuestas como si se tratara de sentencias únicas.





El flujo de un programa en lenguaje PASCAL es gobernado por las denominadas estructuras de control. Las dos estructuras básicas son la "secuencial" y la "repetitiva" o cíclica.

evidente que su presencia innecesaria puede conducir a confusiones, como en el siguiente ejemplo:

```
BEGIN
      BEGIN
            READLN(dato1,dato2)
      END;
      BEGIN
            WRITELN('suma=',dato1+dato2);
            WRITELN('resta=',dato1-dato2);
      END
END.
```

Es necesario resaltar de nuevo la obligatoriedad del punto y coma ";" al final de cada sentencia, dada su función de separador. No obstante, en casos como el del ejemplo anterior, puede ser omitido si a continuación va la palabra END. Hay que recordar que END tiene también un carácter de separador de sentencias. En todo caso, la presencia redundante del punto y coma delante del END no conduce a error alguno.
El punto final ".", localizado detrás del último END, es el que señala el final del programa; si en su lugar apareciera un ";", el ordenador aguardaría a una nueva instrucción, dado que el punto y coma denota únicamente el punto de separación entre dos sentencias.

ESTRUCTURAS REPETITIVAS

En condiciones normales, las sentencias se van ejecutando ordenadamente, y una sola vez cada una de ellas, hasta llegar al final del programa. Sin embargo, en muchas ocasiones es necesario que algunas sentencias se ejecuten repetidamente un determinado número de veces, constituyendo lo que se denomina un "bucle". Este tipo de sentencias, denominadas iterativas o repetitivas, se ejecutan bajo el control de una condición impuesta, cuyo cumplimiento o no determina su repetición.
El PASCAL cuenta con tres sentencias iterativas: WHILE/DO, REPEAT/UNTIL y FOR/TO/DO. La primera de ellas tiene el siguiente formato:

WHILE <expresion> DO <sentencia>;

La <expresion>, de tipo booleano, es la que determina el número de veces que se repetirá la <sentencia>; concretamente, ésta se ejecutará siempre que la <expresion> sea cierta (valor lógico TRUE). Hay que prever que la variable de control de la <expresion> modifique su valor de forma apropiada dentro del bucle; en caso contrario, se podría caer en un bucle infinito, sin final, y éste suele ser un error de programación difícil de detectar.
Para comprender mejor el funcionamiento de una estructura repetitiva, veamos el siguiente ejemplo:

```
BEGIN
      READ(n);
      c=1;prod=1;
      WHILE c<=n DO
            BEGIN
                  prod=prod*c;
                  c=c+1
            END;
      WRITE('producto=',prod)
END.
```

Suponiendo previamente declaradas las variables que intervienen, este segmento de programa escribirá el producto de los "n" primeros números naturales o, lo que es lo mismo, el llamado factorial de "n". Después de obtener el dato "n" y de inicializar los valores de "c" y "prod", se comprueba si es cierta (TRUE) la expresión: c<=n. Si es así, se ejecutará la sentencia que sigue al DO. En este caso se trata de una sentencia compuesta que en-

globa a otras dos elementales; técnica ésta muy utilizada como ya se comentó para "engañar" al DO y poder repetir la ejecución de varias sentencias agrupadas. Una vez ejecutada esta sentencia se vuelve a comprobar la condición; si es cierta, se volverá a ejecutar dicha sentencia. Y así sucesivamente hasta que la expresión produzca el valor FALSE, con lo que no se ejecutará la sentencia asociada al DO, sino la siguiente: en este caso el WRITE que escribe el resultado.
Hay que hacer notar que la sentencia del bucle no se ejecutará ninguna vez si la expresión produce el valor lógico FALSE al evaluarla por vez primera.
No ocurre así con la estructura REPEAT/UNTIL. En ella, sea cual fuere el valor de la expresión en la primera pasada, se ejecutará primero la sentencia, para pasar después a evaluar la condición. Esto significa que siempre se ejecutará al menos una vez la sentencia interior al bucle.
El formato de la sentencia que da cuerpo a una estructura de este tipo es la siguiente:

REPEAT<sentencia> UNTIL <expresion>;

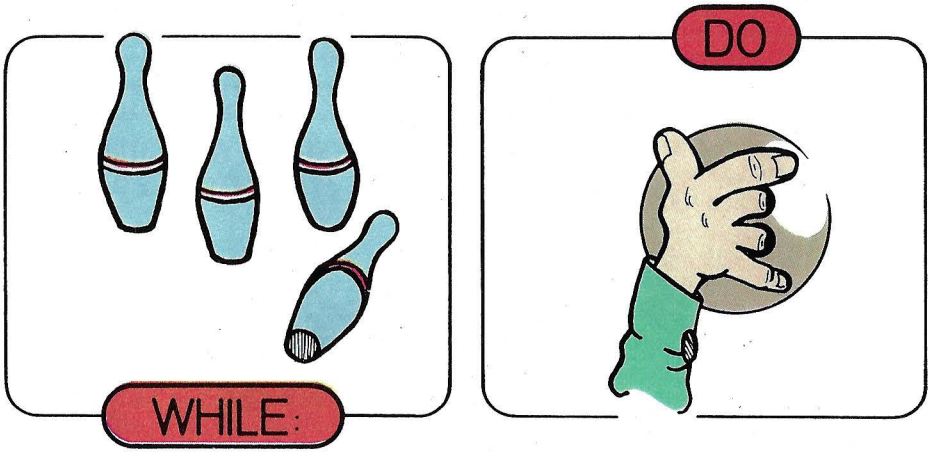
La zona <sentencia> se repetirá (REPEAT) un número indeterminado de veces, hasta que (UNTIL) la <expresion> sea cierta (TRUE). En ese momento se seguirá ejecutando la próxima sentencia del programa, ajena al bucle.
El mismo programa capaz de calcular el producto de los "n" primeros números naturales, se podría codificar con la sentencia REPEAT/UNTIL de la siguiente forma:

```
        BEGIN
              READ(n);
              c=;prod=1;
              REPEAT
                    prod=prod*c,
                    c=c+1
              UNTIL c>n;
WRITE('producto=',prod)
        END.
```

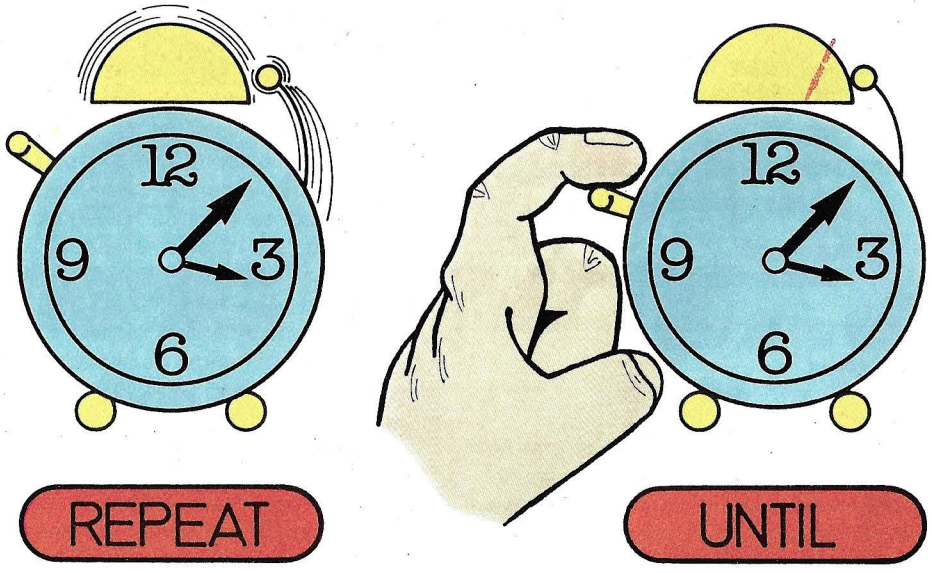
En este caso varía la condición respecto a la del ejemplo que utiliza la sentencia WHILE/DO; debido, precisamente, a la diferencia entre ambas estructuras en cuanto a ejecutar una vez al menos las sentencias del bucle. Otra de las divergencias entre ambas estructuras reside en que ahora no es necesario introducir en un bloque BEGIN/END a todas las sentencias que constituyen el bucle, ya que éstas quedan delimitadas por las palabras REPEAT y UNTIL.
Queda por ver aún una tercera estructura repetitiva, la plasmada por la sentencia FOR/TO/DO. Su principal diferencia con las dos anteriores reside en la funcionalidad de la condición. En las estructuras WHILE/DO y REPEAT/UNTIL, el bucle o conjunto de sentencias se ejecuta reiteradamente hasta que se verifique la condición impuesta, instante en el que se abandonará el bucle. Sin embargo, en el caso de FOR/TO/DO, se repetirá el bucle tantas veces como valores intermedios haya entre los valores inicial y final de la variable de control del bucle. El formato es el siguiente:

FOR <variable de control>:=<variable o valor inicial>
TO <variable o valor final>DO<sentencia>;

En esta sentencia, tanto la variable de control como sus valores inicial y final, deben ser del mismo tipo y, específicamente, de un tipo ordinal: INTEGER, CHAR, o cualquier otro tipo ordinal declarado por el usuario, como los enumerativos y los subrango, que se verán más adelante. La variable de control no puede ser nunca del tipo REAL.



La sentencia WHILE/DO da pie a una estructura repetitiva. En ella se repite una acción determinada mientras sea cierta la condición impuesta.

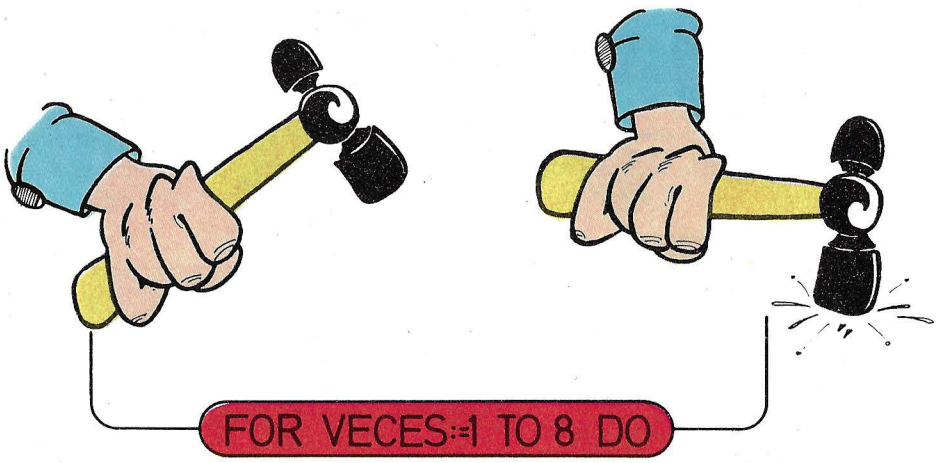


La acción encerrada entre las palabras REPEAT y UNTIL, se ejecutará repetidamente hasta que se verifique la condición impuesta.

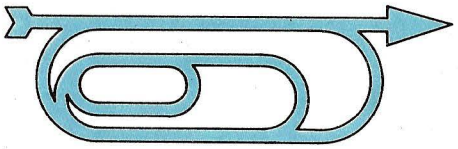
TABLA DE COMANDOS-PASCAL

COMANDO	COMETIDO	OBSERVACIONES
WHILE <expr.> DO <sent.>;	Ejecución de un bucle mientras es cierta la expresión.	Cuerpo del programa
REPEAT <sent.> ;<sent.>;... UNTIL <expr.>;	Ejecución de un bucle hasta que sea cierta la expresión.	Cuerpo del programa
FOR <cont.>:=<ini.> TO<fin.> DO <sent.>;	Ejecución de un bucle según los valores ascendentes de una variable.	Cuerpo del programa
FOR <cont.>:=<ini.>DOWNTO <fin.>DO<sent.>;	Ejecución de un bucle según los valores descendentes de una variable.	Cuerpo del programa
BEGIN <sent.> ;<sent.>;... END;	Agrupación de varias sentencias en una única compuesta.	Cuerpo del programa

NOTA: <expr.>: expresión de tipo booleano. <sent.>: sentencia ejecutable. <cont.>: variable de control del bucle. <ini.>: expresión que da el valor inicial a la variable de control. <fin.>: expresión que indica el valor final de la variable de control.



La sentencia FOR/TO/DO repite la acción un determinado número de veces, de acuerdo a la sucesión de valores que adopta la variable de control.



El anidamiento de sentencias de control permite crear estructuras de mayor complejidad.

El valor de esta variable de control no debe ser alterado por las sentencias del bucle. Los valores inicial y final pueden ser incluso expresiones del mismo tipo que la variable de control; expresiones que serán evaluadas antes de entrar en el bucle.
El funcionamiento de la estructura FOR/TO/DO es el siguiente. En principio, se asigna a la variable de control el valor inicial y se chequea si ya se ha llegado al valor final, en cuyo caso no se ejecutaría el bucle ninguna vez. Si no alcanza aún el valor final, se ejecutará el bucle y, seguidamente, se incrementará el valor de la variable de control antes de ser comprobado de nuevo. Este proceso se repite hasta que la variable de control adopta el valor final, pasando entonces a ejecutar la próxima sentencia del programa.
El ejemplo que sigue dibujará un histograma, o diagrama de barras, con tantos asteriscos por barra como indique el valor de "número", introducido a través del teclado:

```
PROGRAM asteriscos (INPUT,OUTPUT);
VAR contador,numero,barra,numbarras:INTEGER;
BEGIN
      WRITELN('Cuantas barras?');
      READLN(numbarras);
      FOR barra:=1 TO numbarras DO
      BEGIN
          READLN(numero);
          FOR contador:=1 TO numero DO
              WRITE('*');
          WRITELN
      END
END.
```

Como quiera que el bucle más interno está formado por una única sentencia, no es necesario incluirla en un bloque BEGIN/END; no obstante, sí hay que hacerlo con el bucle más externo que consta de más de una sentencia.
La variable de control asociada a esta sentencia puede evolucionar con valores ascendentes o descendentes. Para hacer que "cuente hacia atrás", sólo hay que sustituir la palabra TO por DOWNTO:

```
PROGRAM asteriscos (INPUT,OUTPUT);
VAR contador,numero,barra,numbarras:INTEGER;
BEGIN
    WRITELN('Cuantas barras?');
    READLN(numbarras);
    FOR barra:=numbarras DOWNTO 1 DO
      BEGIN
          READLN(numero);
          FOR contador:=numero DOWNTO 1 DO
          WRITE('*');
          WRITELN
      END
END.
```

SENTENCIAS ANIDADAS

Para tener un control completo del flujo del programa, se utilizan las sentencias de repetición y las sentencias de selección, que más adelante serán objeto de estudio. Todas ellas incluyen otras sentencias de control gobernadas por las que la contienen. De esta forma se pueden crear estructuras de cualquier nivel de complejidad, "anidando" unas sentencias en otras. El tratamiento que se hace de las sentencias compuestas, por parte del usuario, coincide con el de las sentencias simples. Esta característica otorga al PASCAL una gran legibilidad, facilitando la comprensión de los programas de un simple vistazo.

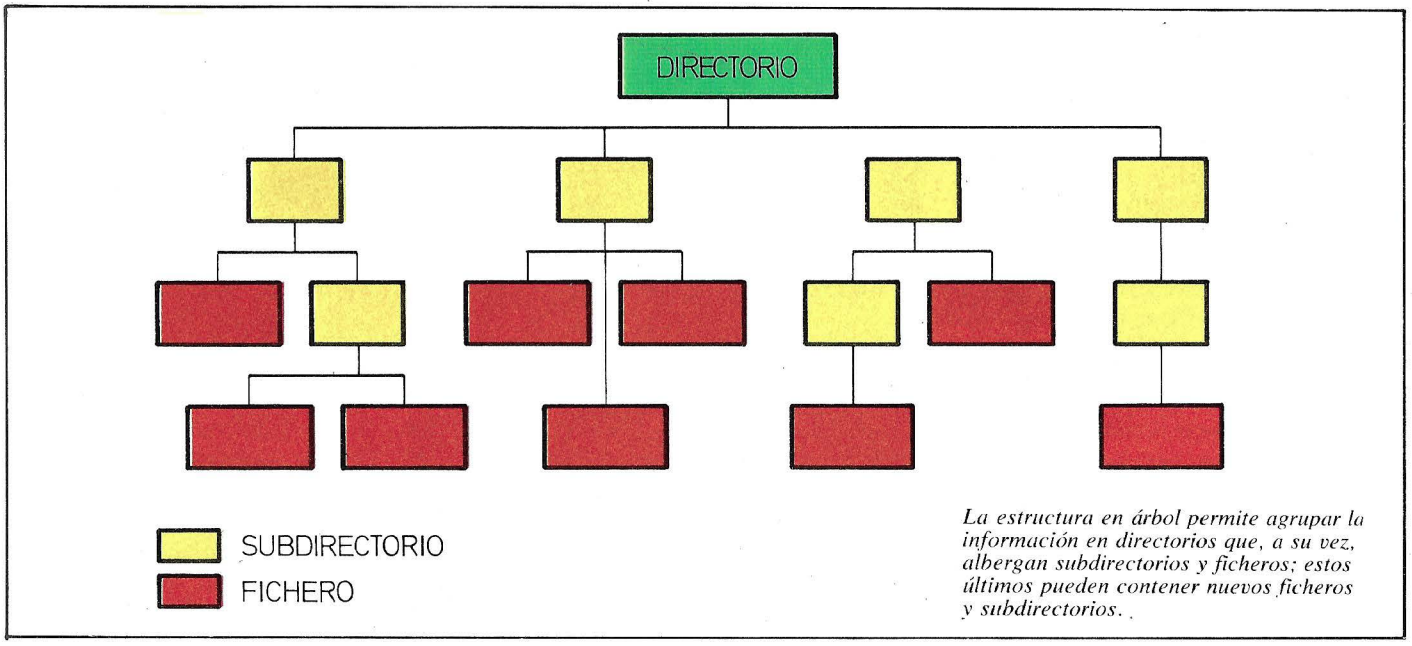
MS/DOS (6)

Características especiales del DOS 2.0

La aparición de discos rígidos con capacidades del orden de decenas de Megabytes ha tenido mucho que ver en la reorientación del desarrollo del sistema operativo MS/DOS. Esta es una incidencia ostensible a partir de la versión DOS 2.0.
La disponibilidad de un gran volumen de datos residentes en un disco rígido tipo Winchester, ha hecho que las nuevas versiones del MS/DOS incluyan poderosas y flexibles herramientas para el control y el manejo de estos datos. Estas eran herramientas no contempladas en versiones anteriores, ya que al realizarse el almacenamiento principalmente en disquetes, no parecía razonable articular un complejo proceso de tratamiento para un volumen de datos relativamente pequeño.
Entre las novedades que aporta la versión 2.00 del MS/DOS en lo relativo al control del flujo de datos, destacan la organización de datos en directorios y subdirectorios, el direccionamiento de las entradas y salidas de un programa, la concatenación de programas empleando como ficheros de entrada de datos los ficheros de salida del programa ejecutado anteriormente y, finalmente, el cambio de consola del sistema. A continuación se describen individualmente cada uno de estos puntos.

LA ESTRUCTURA ARBORESCENTE DE DATOS

El almacenamiento de datos en archivos residentes en disco (ya sean discos rígidos o flexibles) no está exenta de problemas. La localización de un determinado elemento de información puede que no sea tan clara como en un principio cabría pensar, debido a la cantidad de información almacenada y a la ambigüedad de ésta, sobre todo en sistemas que disponen de un disco rígido de gran capacidad.
Para resolver esta complicación, el MS/DOS versión 2.00 permite crear dentro de un mismo disco distintos directorios que pueden contener archivos o, a su vez, nuevos directorios (también llamados subdirectorios). Así pues, ahora se puede crear una estructura lógica con la información y almacenar los datos que se refieren a un tema específico en su propio directorio. Ello contribuirá, sin lugar a dudas, a disponer de una estructura clara y ordenada de los datos.
Cualquier disco contiene un directorio maestro o raíz, creado a la hora de formatear el disco, del cual pueden derivar nuevos subdirectorios capaces de acoger ficheros en grupos completamente estancos. Ello da nacimiento a una estructura de información en forma de árbol.



La estructura en árbol permite agrupar la información en directorios que, a su vez, albergan subdirectorios y ficheros; estos últimos pueden contener nuevos ficheros y subdirectorios.

La inclusión de un nuevo directorio se realiza mediante el comando MKDIR (MD en forma abreviada); su activación crea un directorio vacío. Una vez creado el directorio, éste se puede ir rellenando con archivos o subdirectorios cuyos nombres pueden ser iguales a los nombres ya existentes en otros directorios. Al respecto, hay que precisar que el nombre de cada fichero está determinado por su nombre y, además, por lo nombres de los directorios que lo contienen.

Dado que un disco puede incluir numerosos subdirectorios, organizados en una estructura de tipo árbol, hay que establecer un criterio para poder indicar en qué rama de la estructura se desea actuar. Para ello se define la noción de camino (path) como «la ruta que habría que seguir a través de los distintos directorios para llegar a la zona deseada».

Suponga que nos encontramos ante un disco con su directorio raíz que contiene, entre otros, un directorio llamado COMARCA; este último contiene a su vez otro llamado HOBBITON, y dentro de este último hay un fichero de nombre BOLSON. Para describir el camino que conduce hasta el fichero BOLSON habrá que proceder como sigue:

— Empezando en el directorio raíz, buscar el subdirectorio COMARCA.
— Localizar el subdirectorio de COMARCA cuyo nombre es HOBBITON.
— Por último, buscar dentro del subdirectorio HOBBITON el fichero BOLSON.

La separación entre los distintos nombres de los subdirectorios y los ficheros se hace mediante una barra inclinada o «slash», por lo que la representación del camino a seguir coincidirá con:

/COMARCA / ALTCAMP / VALLS

La definición de los caminos puede ser absoluta si se parte del directorio raíz, o relativa si se comienza desde alguna rama del árbol. Un ejemplo de direccionamiento absoluto sería el anteriormente descrito, ya que el directorio raíz se indica al empezar el camino con una barra inclinada. Por el contrario, un ejemplo de direccionamiento relativo sería:

ALTCAMP / BAG-END / ALIO

El trabajo continuo con caminos puede resultar molesto si el trabajo se desarrolla en un subdirectorio determinado, ya que los nombres de los ficheros sobre los que se va actuar tendrían que llevar siempre el prefijo indicativo del camino para llegar hasta ellos. Este inconveniente puede obviarse, ya que se puede indicar al MS/DOS cuál es el directorio sobre el que se desea actuar y éste nos posicionará sobre dicho directorio. De esta forma, una vez ubicados en el directorio elegido, ya es totalmente innecesario indicar el camino para llegar a él.

El comando que realiza la mencionada función es CHDIR o, más abreviadamente, CD. Por ejemplo, para hacer que el subdirectorio «/COMARCA/ALTCAMP» sea el directorio sobre el cual se desea operar, bastará con escribir el comando:

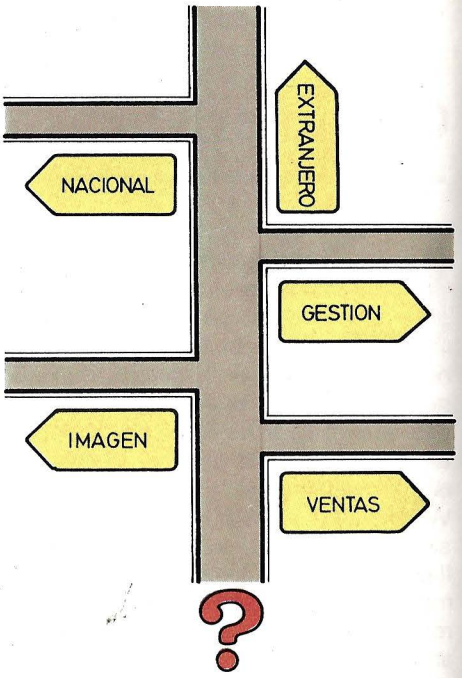
CHIR / COMARCA / ALTCAMP

Asimismo, también es posible cambiar de directorio si se dispone de varias unidades de disco y se desea pasar de un subdirectorio que está en un disco a otro subdirectorio localizado en otro disco. Para ello, es suficiente con referenciar el disco implicado antes de apelar al directorio. Eso es:

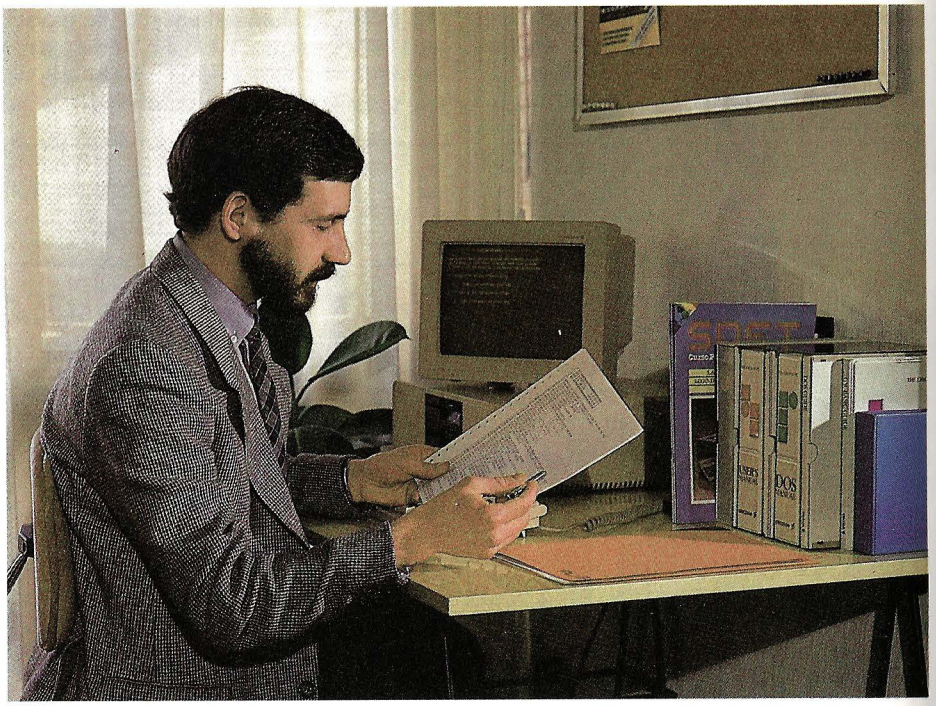
CHIR A: / MORDOR / MONTE / MONTE-DESTINO / ANILLO

Otros comandos relacionados con la estructura en árbol de la información son:

— RMDIR o RD: para borrar un directorio que previamente se ha vaciado de ficheros y subdirectorios.
— TREE: muestra una lista de todas las ramas del árbol que cuelga del directorio



La tarea de localizar la información resulta más racional si ésta queda integrada en una estructura arborescente.



La versión 2.00 del MS|DOS brinda al usuario la posibilidad de organizar la información en estructuras lógicas de tipo arborescente. Esta es una técnica que se revela de suma eficacia a la hora de organizar racionalmente grandes volúmenes de información.

raíz, para todas las unidades de disco conectadas.
— PWD: indica cuál es el directorio sobre el que está posicionado el sistema.
— PATH: define los caminos a través de los que han de buscarse los programas.

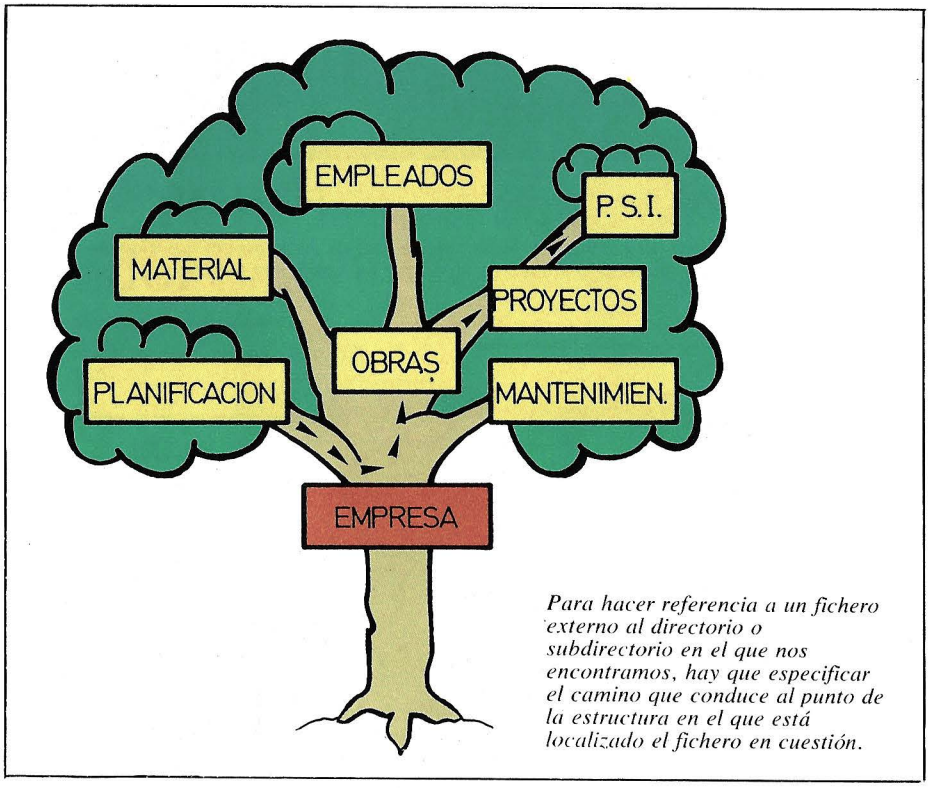
OTROS MODOS DE COMUNICACION

Normalmente, cualquier programa exige la introducción de los datos necesarios para su ejecución a través del teclado y, a su vez, hace patentes los resultados visualizados en la pantalla asociada al sistema. Bajo la versión 2.00 del MS/DOS, la exposición exterior sigue siendo cierta, si bien tiende a ser más general. El programa lee información de una entrada estándar y plasma los resultados en una salida estándar, teniendo ambas la particularidad de que no necesariamente tienen por qué coincidir con el teclado y la pantalla, sino que pueden asignarse otros dispositivos a estas entradas y salidas.
En general, el redireccionamiento de las entradas y salidas estándar se realiza con ficheros o dispositivos que memoricen la información el tiempo suficiente como para que ésta pueda ser revisada o actualizada en cualquier instante.
Suponga que nos encontramos ante un programa que proyecta los resultados de salida por pantalla y se desea que esta información quede archivada. Para lograrlo, será preciso corregir el programa para que los resultados queden ahora registrados en un fichero o se escriban a través de una impresora. Sin embargo, utilizando la posibilidad de redireccionar la salida, se puede conseguir que los resultados queden grabados en un fichero sin necesidad de alterar el programa. El método consiste, simplemente, en especificar el nombre del fichero en el que se desean grabar los resultados a continuación de nombre del programa; por ejemplo:

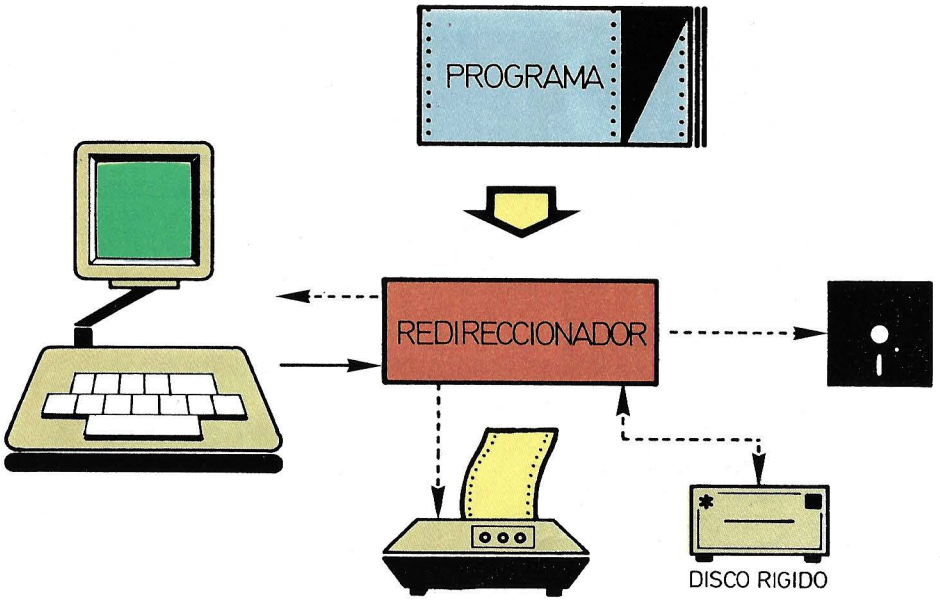
NOMINA > ENERO

El signo > se reserva para indicar a dónde se desea dirigir la salida. Por el contrario, el signo < se utiliza para indicar de dónde procede la entrada de datos.
Igualmente, se puede conseguir que un programa reciba los datos procedentes de un fichero en vez de recibirlos desde el teclado:

NOMINA < DATOS



Para hacer referencia a un fichero externo al directorio o subdirectorio en el que nos encontramos, hay que especificar el camino que conduce al punto de la estructura en el que está localizado el fichero en cuestión.

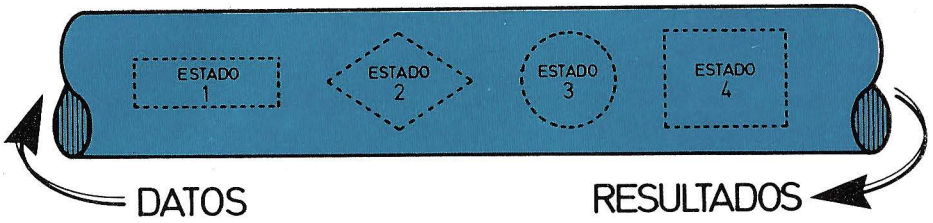


Por medio del MS/DOS versión 2.00, es posible modificar las entradas y salidas de un programa, sin que por ello haya que alterar el contenido del mismo.

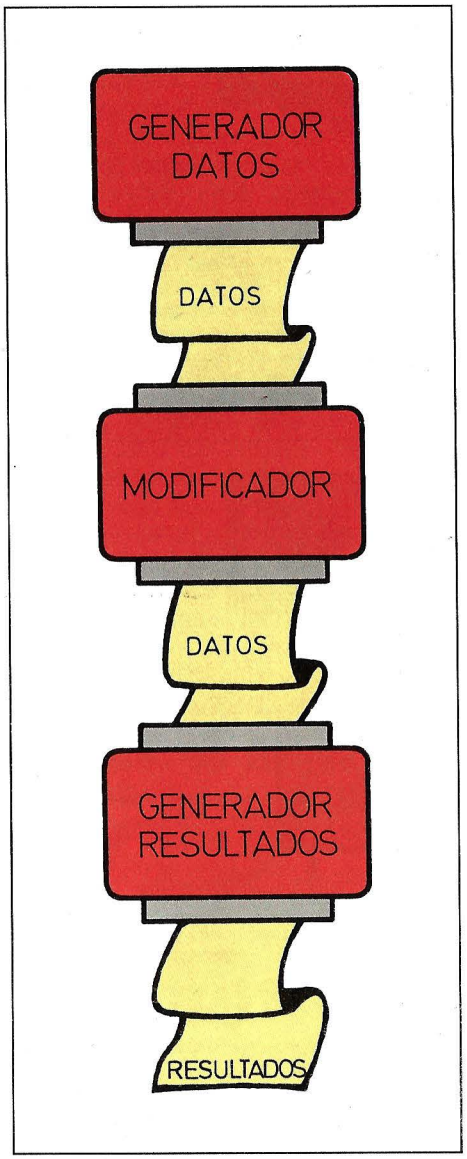
Y también es posible redireccionar la entrada y salida al mismo tiempo:

NOMINA > ENERO < DATOS

La razón principal para redireccionar la salida es almacenar la información resultante para una posterior consulta o revi-



Concatenando programas que actúen sobre los resultados de otros programas ejecutados anteriormente, puede construirse una cadena en la que los datos van pasando por sucesivos estados de transformación.



Dentro de una cadena de programas o "pipeline" cabe distinguir tres tipos de programas: generador de datos, modificador de datos y generador de resultados.

sión; mientras que el redireccionamiento de la entrada suele obedecer al hecho de evitar la necesidad de introducir los mismos datos si éstos tienen siempre un mismo formato.

COMO CONSTRUIR UN «PROGRAMADUCTO»

Los programas manipulan datos en orden a resolver la tarea que se les ha encomendado. Y es probable, incluso, que un programa utilice como datos de entrada los datos de salida generados por un programa ejecutado con anterioridad. Con lo visto en el apartado anterior, se deduce fácilmente que el problema quedaría resuelto ejecutando el primer programa y redireccionando la salida a un fichero; a partir de ahí, sólo habrá que ejecutar a continuación el segundo programa, redireccionando su entrada al fichero creado como salida del primer programa. Esto es:

```
ESCRIBIR > TEXTO
CORREGIR < TEXTO
```

No obstante, el MS/DOS versión 2.00 ofrece una variante aun más sencilla para llevar a cabo este proceso, ya que puede encargarse de crear el fichero que sirve para intercambiar los datos y pasarlos de un programa a otro. Esto es lo que se denomina cadena de programas o «pipeline» en la jerga del sistema operativo. Para crear un «pipeline» es suficiente con escribir los nombres de los programas separados por una barra vertical:

```
ESCRIBIR | CORREGIR
```

El formato de esta expresión puede hacer pensar que no se utiliza ningún fichero intermedio para pasar la información de un programa a otro. Ello no es cierto, ya que el sistema operativo se encarga de crear un fichero temporal en el cual el primer programa escribe sus resultados y del que el segundo programa lee sus datos; fichero que se borra automáticamente al finalizar la ejecución del segundo programa.

Hay una diferencia obvia entre los programas que ocupan la primera posición, la intermedia y la última dentro de esta cadena o «pipeline». El primer programa ha de ser necesariamente un generador de datos capaz de alimentar a los que vienen a continuación. Los programas emplazados en medio de la cadena deben consumir los datos generados por el programa anterior y, a su vez, tienen que generar otros datos para que sean utilizados por los programas que siguen. Y llegamos al extremo de la cadena. Sin lugar a dudas, el último programa debe ser un consumidor de datos nutrido por la información de salida de los programas que le preceden.

EL CAMBIO DE CONSOLA

Los dos últimos métodos útiles para direccionar la entrada y salida de un programa, actúan mientras éste se encuentre en ejecución; una vez concluida la ejecución, puede recurrirse de nuevo a las vías tradicionales: el teclado y la pantalla. No obstante, si se utiliza el comando CTTY, se puede conseguir que sea otro dispositivo quien haga las funciones de pantalla y teclado.

Este dispositivo debe ser capaz de actuar como un teclado y una pantalla a la vez; tal condicionante descarta a cualquier dispositivo para almacenamiento de información, como son los discos. Las condiciones exigidas las cumple, normalmente, un dispositivo de comunicaciones remotas, el cual puede pasar a constituir la consola del sistema:

```
CTTY    AUX
```

De esta forma se puede conseguir un control remoto sobre el ordenador, quedando abierto el acceso al mismo desde un punto alejado al emplazamiento del ordenador, ya sea por vía telefónica o a través de cualquier otra línea de comunicación.

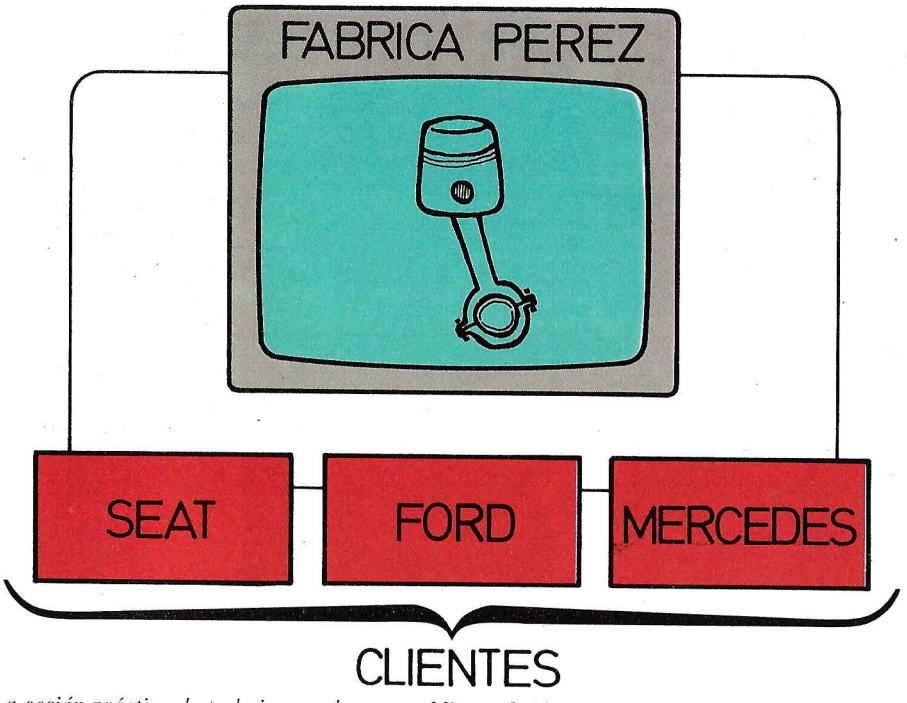
Microsoft Chart (y 2)

Creación con ordenador

Al igual que se ha hecho con todas las aplicaciones estudiadas a lo largo de la obra, vamos a describir una típica sesión de trabajo con el programa MICROSOFT CHART.

SUPUESTO DE TRABAJO

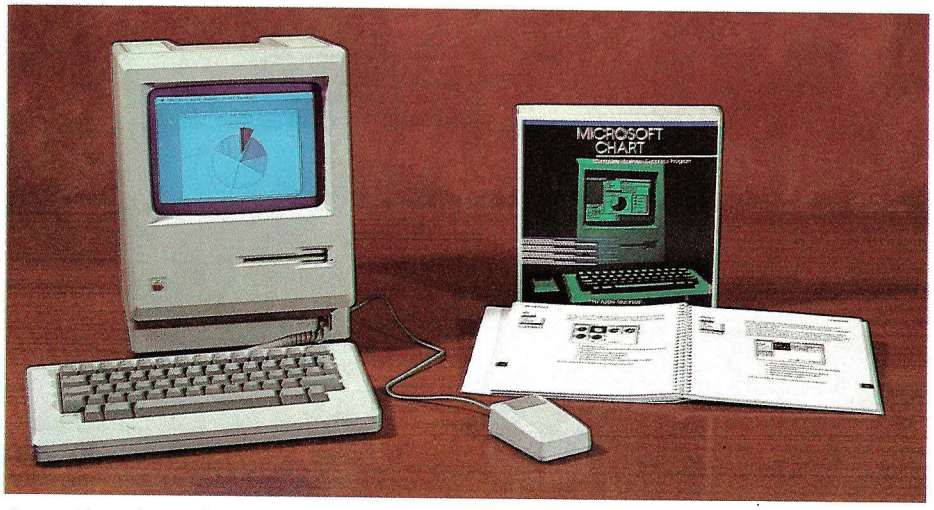
El supuesto usuario del MICROSOFT CHART va a ser, en nuestro ejemplo, el departamento de facturación de una fábrica. El fenómeno que se desea estudiar es la evolución mensual de la facturación; por un lado a nivel global, es decir considerando el importe facturado a todos los clientes, y por otro lado un detalle de la facturación por cada cliente.
Supondremos que la fábrica en cuestión se dedica a fabricar piezas para motores que posteriormente vende a tres únicos clientes: SEAT, FORD y MERCEDES. Los respectivos volúmenes de facturación, medidos en miles de pesetas, se relacionan en la tabla abjunta.



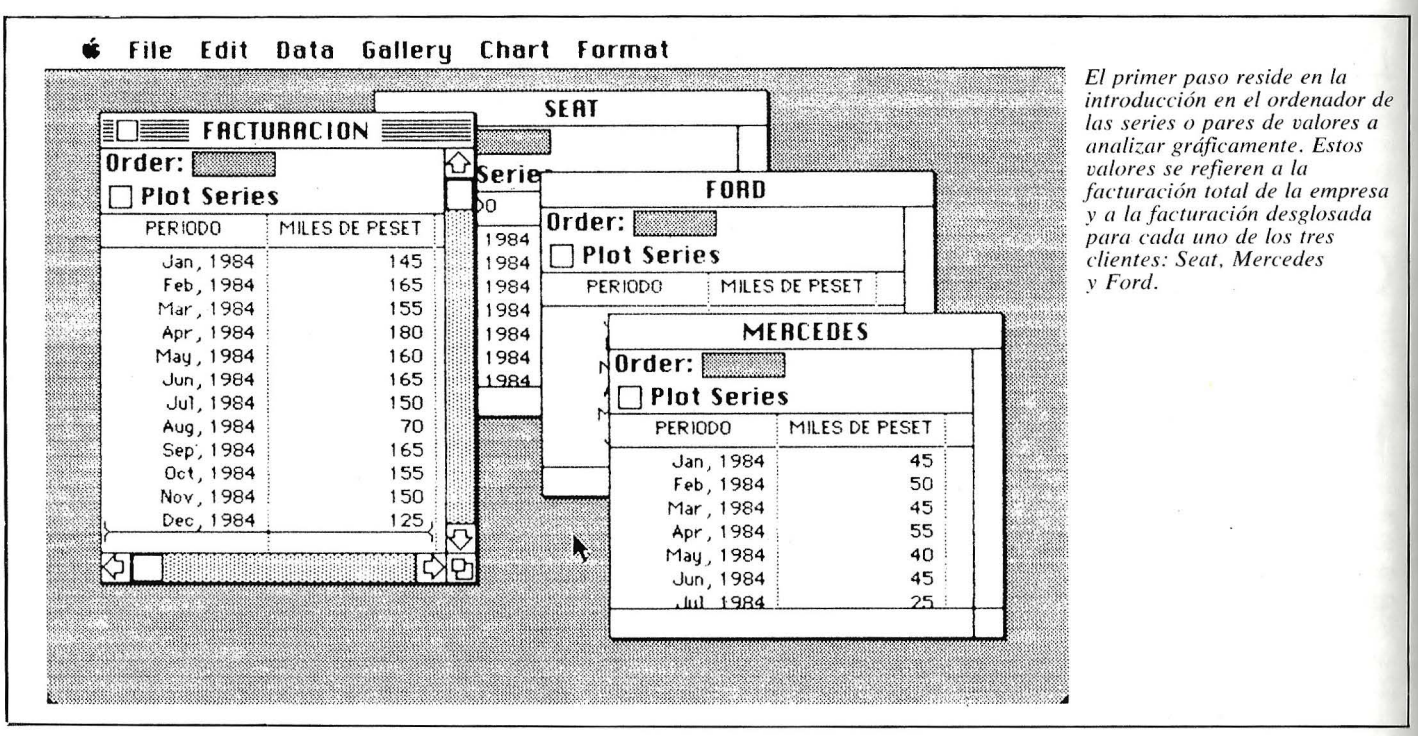
La sesión práctica de trabajo con el paquete Microsoft Chart, contempla la actividad económica de una supuesta empresa dedicada a la fabricación de piezas para motores de automóvil.

DIAGRAMA SIMPLE

La sesión de trabajo comenzará cargando el programa MICROSOFT CHART en la memoria del ordenador. De inmediato, aparecerá una ventana en la pantalla en la que se introducirán los datos a representar. La ventana, que aparecerá con la denominación «New series» (nueva serie),



La sesión práctica de trabajo se ha realizado en base a la versión del paquete gráfico Microsoft Chart creada para el ordenador Apple Macintosh.

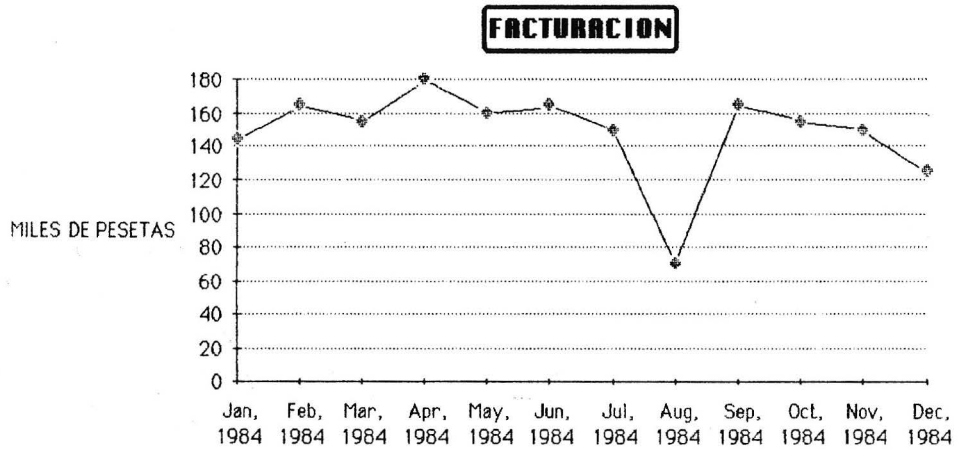


El primer paso reside en la introducción en el ordenador de las series o pares de valores a analizar gráficamente. Estos valores se refieren a la facturación total de la empresa y a la facturación desglosada para cada uno de los tres clientes: Seat, Mercedes y Ford.

MES / CLIENTE	I 84	II 84	III 84	IV 84	V 84	VI 84	VII 84	VIII 84	IX 84	X 84	XI 84	XII 84	TOTAL 1984
SEAT	75	80	70	85	85	90	95	55	70	85	90	80	960
FORD	25	35	40	40	35	30	30	10	50	20	30	35	380
MERCEDES	45	50	45	55	40	45	25	5	45	50	30	10	445
TOTAL	145	165	155	180	160	165	150	70	165	155	150	125	1785

Facturación en miles de pesetas.

Tabla de valores que refleja el supuesto práctico a analizar por medio del paquete Microsoft Chart.



El primer diagrama simple que podemos obtener es el representativo de la facturación total de la empresa a lo largo de los doce meses del año.

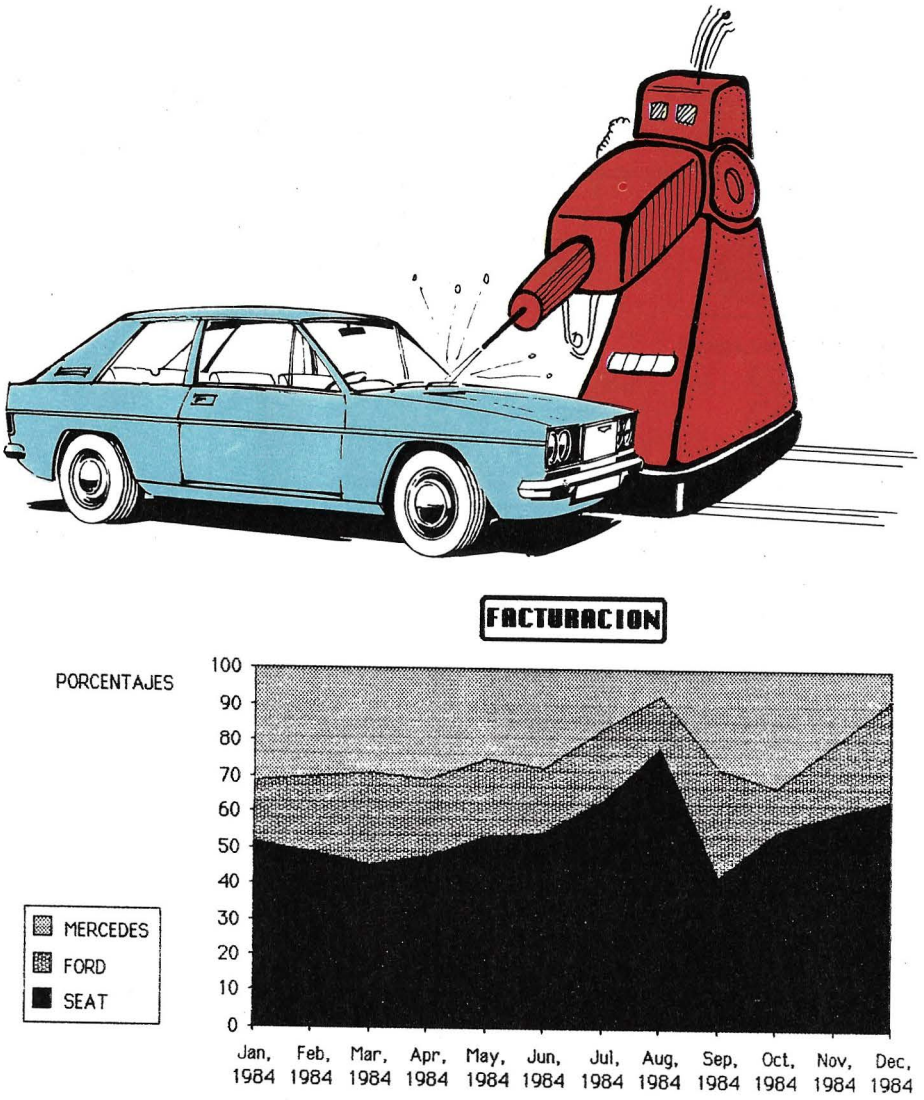
presenta dos columnas denominadas X e Y.

Como nuestro primer objetivo es obtener en forma gráfica la evolución global de la facturación de la fábrica, el tipo de diagrama más apropiado es el de línea; por lo tanto, en la columna Y se introducirá la suma total de la facturación en cada uno de los doce meses que vamos a estudiar, y en la columna X habrá que escribir la denominación que daremos a cada uno de los meses en el gráfico. Esta última tarea que puede realizar la aplicación de forma automática, sin más que invocar al menú DATA. Este ofrece cuatro posibilidades: SEQUENTIAL, DATE, TEXT y NUMERIC. La primera sirve para la numeración secuencial y es la utilizada por defecto; la segunda para la numeración automática de fecha; la tercera para introducir textos como marcas de categorías y la cuarta para incluir marcas de tipo numérico.

En nuestro caso, la opción ideal sería DATE. Al seleccionarla aparecerá una nueva ventana en la que decidiremos que las categorías del diagrama son fechas, con formato mes/año y con secuencia unitaria.

Desde luego, y tal como se observa en las figuras, las indicaciones aparecerán en inglés (Jan, 1984; Feb, 1984...). Si deseamos que figuren en castellano, habrá que decidirse por la opción TEXT, e introducir manualmente los textos correspondien-

tes a la denominación de cada mes; esto es: ENE-84, FEB-84, MAR-84...
Ahora hay que introducir ya los valores de Y. Para ello, teclearemos primero el número 145 y tras ello pulsaremos la tecla ENTER o RETURN. A continuación teclearemos 165, 155, etc., hasta completar la facturación total de los doce meses. De forma automática, el programa irá presentando la secuencia mes/año adecuada en la columna X.
Al terminar la introducción de los datos, tendremos doce pares de valores representables gráficamente. Su aspecto es el que muestra la serie denominada FACTURACION que se reproduce en la figura adjunta.
A continuación, sin más que seleccionar la opción GALLERY, podremos determinar el diagrama apropiado, que en nuestro caso será el de líneas. Tras ello elegiremos el menú CHART, y mediante su opción ADD LEGEND incluiremos en el gráfico textos explicativos; por ejemplo «FACTURACION» y «MILES DE PESETAS». En ambos casos, su diseño y colocación está apoyada por opciones que brindan los menús. Para terminar, en el caso de que así lo deseáramos, podríamos ordenar la confección de una copia del gráfico por impresora o su almacenamiento en un soporte de memoria permanente para su posterior uso.



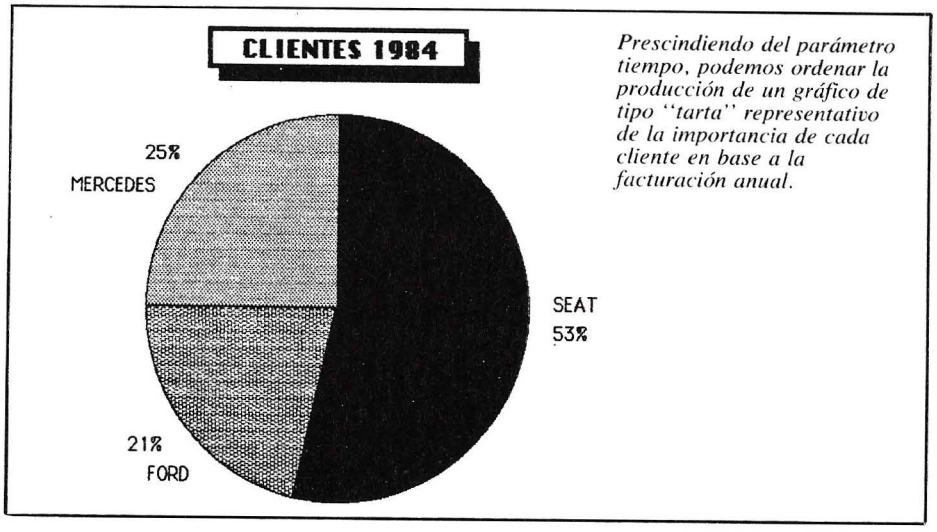
Para ilustrar gráficamente la importancia relativa de cada cliente a lo largo de los sucesivos meses, podemos recurrir a un diagrama de superficies. Su aspecto coincidirá con el ilustrado.

DIAGRAMAS COMPUESTOS

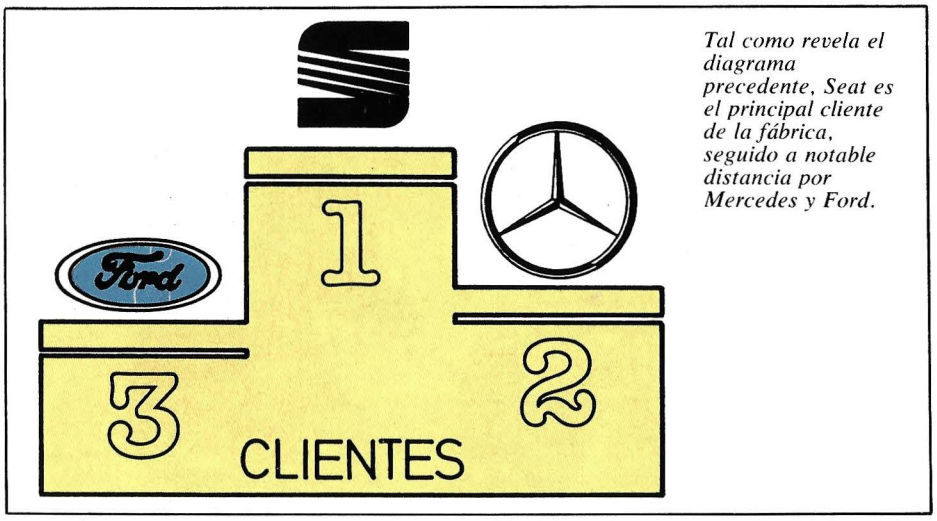
Al diagrama descrito en el párrafo anterior se le puede denominar diagrama simple, puesto que en él tan sólo se estudia un fenómeno en distintos tiempos. En cambio, en los diagramas que van a ocuparnos en los próximos párrafos, se estudian simultáneamente varios fenómenos, uno por cada cliente; por lo tanto, hay que hablar ahora de diagramas compuestos. Para estudiar la facturación de la fábrica cliente por cliente, vamos a analizar gráficamente los aspectos que se detallan a continuación.

- *Importancia relativa de clientes*

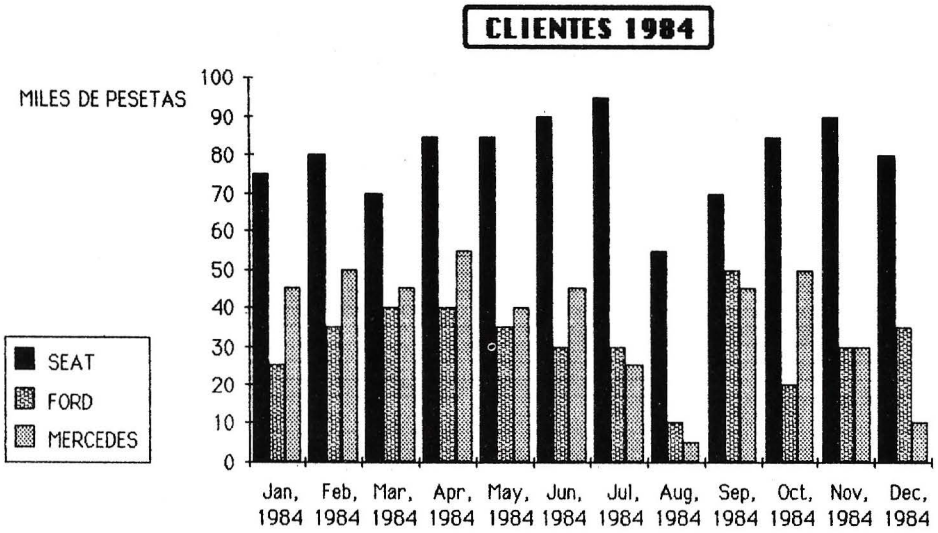
Para describir gráficamente que tan importante es un cliente comparado con los restantes, a lo largo de los doce meses



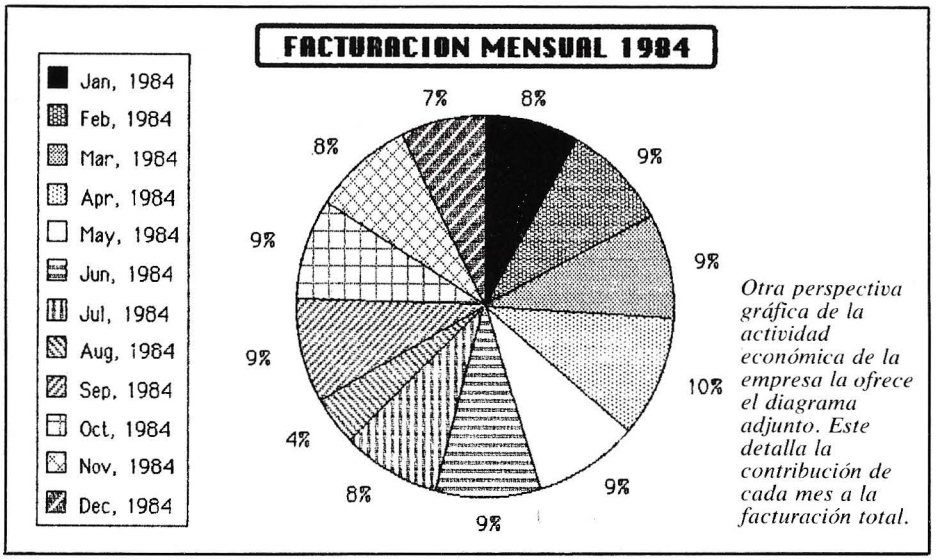
Prescindiendo del parámetro tiempo, podemos ordenar la producción de un gráfico de tipo "tarta" representativo de la importancia de cada cliente en base a la facturación anual.



Tal como revela el diagrama precedente, Seat es el principal cliente de la fábrica, seguido a notable distancia por Mercedes y Ford.



A la hora de comparar la actividad económica con cada uno de los tres clientes, resulta muy adecuado obtener un diagrama de barras detallado, semejante al que ilustra la figura.



Otra perspectiva gráfica de la actividad económica de la empresa la ofrece el diagrama adjunto. Este detalla la contribución de cada mes a la facturación total.

estudiados, el diagrama más apropiado es el de superficies. Este permitirá comparar el porcentaje de superficie que le corresponde a cada cliente en cada mes. Sin más que echar una ojeada al gráfico de la figura correspondiente, se puede observar que el cliente SEAT es el más importante de la fábrica, con notable diferencia sobre los otros dos, mientras que MERCEDES es mejor que FORD aunque por cada diferencia y no en todos los meses.

- *Facturación anual a clientes*

Si prescindimos de parámetro tiempo, esdecir, si lo que pretendemos resaltar es la importancia de los clientes pero no mes a mes sino respecto a la facturación anual en un único diagrama, sin duda el tipo de gráfico más apropiado es el de tarta. En él podremos ver cómodamente la porción de tarta que corresponde a cada cliente. En el gráfico de la figura, adaptado al ejemplo que nos ocupa, se aprecia como SEAT recibe más de la mitad de la «tarta», mientras que FORD y MERCEDES se reparten el resto.

- *Comparación anual de clientes*

En los dos diagramas anteriores obteníamos una representación significativa de la facturación de la empresa a cada uno de sus clientes, aunque ambos hacían referencia a valores porcentuales.

Ahora vamos a obtener un gráfico ilustrativo de la misma facturación, pero detallada en miles de pesetas. En estas condiciones, el diagrama que más resaltará las diferencias entre los tres clientes será el de multibarras. El eje vertical medirá las pesetas facturadas y en el horizontal se acumularán tres barras, una por cada cliente, tan altas como sea la facturación de cada mes.

En estos tres últimos ejemplos se ha prestado mayor atención a los motivos que determinan el tipo de diagrama más apropiado y a su interpretación, que a los distintos pasos necesarios para obtener los gráficos con el programa MICROSOFT CHART.

Los pasos a seguir están plenamente apoyados por los diversos menús del paquete de aplicación. Estos, aun siendo generalizables, presentan matices característicos dependiendo de la versión del paquete MICROSOFT CHART de que se trate. Al respecto, cabe precisar que el texto y los ejemplos se han confeccionado en base a la versión creada para el ordenador Apple MacIntosh.

Tipos de variables

Representación de datos en el BASIC

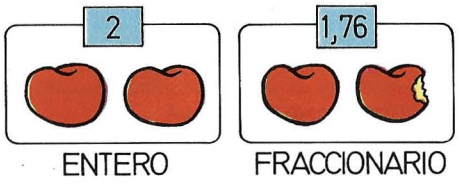
Las magnitudes medibles se expresan en unidades. Así, para longitud se emplea el metro, para masa el gramo, etc. Sin embargo, es corriente acudir a múltiplos o submúltiplos cuando la cantidades son muy grandes o muy pequeñas. Esto es lo que sucede, por ejemplo, con los metros. Nadie indicaría la distancia que separa Madrid de La Coruña en metros. Por lo general estas distancias grandes se suelen expresar en kilómetros. De la misma forma, el tamaño de una moneda se da en centímetros o incluso en milímetros. En aplicaciones matemáticas, y científicas en general, se emplean frecuentemente cantidades muy pequeñas o muy grandes. En ese caso se recurre a una representación exponencial, evitándose de este modo la necesidad de escribir un gran número de cifras. Veamos un ejemplo:

$12000000 = 1{,}2 \times 10^{7}$
$0{,}000002 = 2 \times 10^{-6}$

Como se observa, la representación exponencial evita la exigencia de escribir una larga ristra de cifras (especialmente cuando éstas no son más que ceros). Además, puede suceder que aunque las restantes cifras no coincidan con el cero, se opte por realizar una aproximación. Ello será válido cuando el error cometido no sea significativo. Por ejemplo:

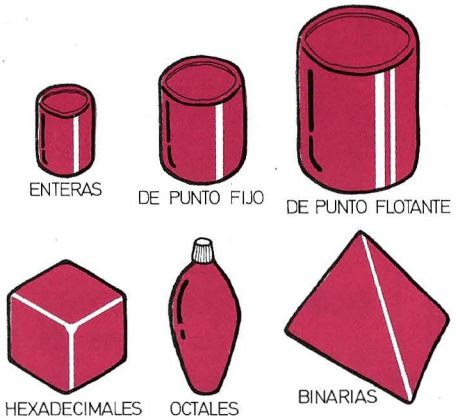
$1000002 \approx 10^{6}$

Desde luego, con esta aproximación se comete un error. De todas formas, a veces es más práctico trabajar con aproximaciones si el error puede considerarse despreciable o mínimo.
Una de las posibilidades más fáciles de explotar en un ordenador es la capacidad que éste tiene para realizar cálculos numéricos. Los datos pertinentes, o bien se



En un primer eslabón, el BASIC debe contemplar la posibilidad de operar tanto con datos enteros como fraccionarios.

encuentran ya incluidos en el propio programa o son introducidos durante la ejecución del mismo.
En cualquier cálculo realizado con el ordenador hay que considerar dos factores básicos: los datos que servirán de materia prima y la forma en la que el ordenador los almacena. Todo ello influirá directamente en la ejecución del cálculo.
Es evidente que la máquina ha de guardar los datos para más tarde operar con ellos. El lugar en el que almacena o "anota" los datos en cuestión no es otro que la memoria; en ella almacena tanto los datos iniciales como los resultados obtenidos.
En este sentido, hay que tener en cuenta que la memoria del ordenador es limitada, de ahí que, en muchos casos, el usuario se vea obligado a economizarla en la medida de lo posible.
Por otra parte, los datos con los que se ha de operar no son siempre semejantes: pueden incluir cifras decimales en mayor o menor cantidad, pueden representar magnitudes muy elevadas o muy reducidas... Estos son criterios que harán necesario un distinto espacio de memoria para su almacenamiento.
Dado que el ordenador no puede conocer a priori la longitud de un número, estos deben almacenarse bajo algún formato estándar. De este modo, es evidente que los datos de poca longitud desaprovecharán un considerable espacio de memoria.
Para solventar estos inconvenientes, el BASIC admite distintos tipos de formatos para el almacenamiento de los datos.



En función del tamaño y del tipo de la constante, el ordenador la almacenará adoptando uno u otro formato entre el repertorio de admisibles.

TIPOS DE REPRESENTACIONES NUMERICAS

El primer caso es aquel en el que el ordenador ha de almacenar un número entero. Para ello se recurre a una representación binaria pura, empleando a tal fin dos bytes de memoria. Teniendo en cuenta que el primer bit se emplea para el signo, el mayor número representable es el 65535.
Aumentando el número de bytes utilizados, se puede conseguir la representación de números mayores. Sin embargo, este método no resulta práctico desde el punto de vista del aprovechamiento idóneo de la memoria del ordenador. En definitiva, o bien se dispone de una gran variedad de tipos de formato o no quedará más alternativa que despilfarrar espacio de memoria cuando haya que almacenar números de reducida magnitud.
También queda pendiente el caso de los

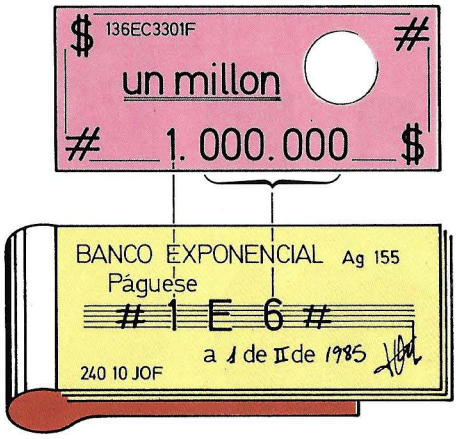
La diversidad de los datos que debe procesar el ordenador, supone precisar distintos formatos para su representación interna y almacenamiento en la memoria de la máquina.

números con parte fraccionaria. ¿Cómo representar este tipo de datos?
Una posible solución reside en adoptar un convenio y fijar la posición de la coma decimal. De este forma, el número de decimales queda fijado también. Esta representación recibe el nombre de "punto fijo" (o coma fija).
Las limitaciones que presenta la representación en punto fijo pueden ser solventadas mediante un nuevo sistema. Este consiste en utilizar la notación científica de la que se habló al principio. Dicha notación consiste en dividir el número en dos partes. Una de ellas coincide con los dígitos más significativos, mientras que la otra da una idea del orden de magnitud de la cantidad, expresada como potencia de 10. El producto de ambas partes dará un resultado equivalente al número a representar.
En el interior de la máquina, estas dos partes se almacenan como "mantisa" y "exponente". La primera aporta los dígitos más representativos, en coma fija. La segunda guarda la potencia de 10 pertinente, a modo de número entero. Esta representación es denominada "punto (o coma) flotante".
Habitualmente, existen dos tipos de representación en coma flotante: simple y doble precisión.
En simple precisión se utilizan cuatro bytes: los tres primeros destinados a la mantisa y el cuarto al exponente. Los bits situados más a la izquierda en cada una de las zonas indican el signo.
La representación en doble precisión emplea siete bytes para la mantisa, y uno sólo para el exponente. Como en el caso anterior, el bit situado más a la izquierda denota el signo de cada parte.

FORMATOS DE ALMACENAMIENTO EN BASIC

En el lenguaje BASIC, los formatos más habituales para el almacenamiento de datos numéricos son los siguientes:
—*Constantes enteras:* se emplean para representar números enteros comprendidos entre −32768 y 32767. No admiten punto decimal.

Ejemplos: 3, 13500, −20000



El formato exponencial permite expresar números de magnitud extrema, muy grandes o muy pequeños, sin tener por ello que utilizar una ingente cantidad de cifras.

—*Constantes de punto fijo:* son números reales en los que se emplea el punto para separar la parte entera de la fraccionaria, adoptando, en consecuencia, la notación anglosajona. No admite más signos de puntuación. Es decir, no se emplean las comas para separar las unidades de millar ni de millón.

Ejemplos: 22.57, 10.0, −12.1, 0.12

—*Constantes de punto flotante:* números positivos o negativos representados en forma exponencial (notación científica). Constan de un número entero o en punto fijo (mantisa) y un entero (exponente). Ambas zonas se separan por medio de la letra E o D. Los valores del exponente no pueden ser mayores de 38 o inferiores a −38.

Ejemplos: 2E3, 3.5 E−6, 7.08 D 14.

Dentro de las constantes de punto flotante cabe distinguir dos tipos: de simple precisión y de doble precisión.
Una constante de simple precisión es toda aquella que satisface los siguientes requisitos:

- Posee hasta siete dígitos de precisión.
- Adopta la forma exponencial utilizando la letra E.
- Incluye un signo de admiración al final (!).

Constante de doble precisión será, por el contrario, la caracterizada por:

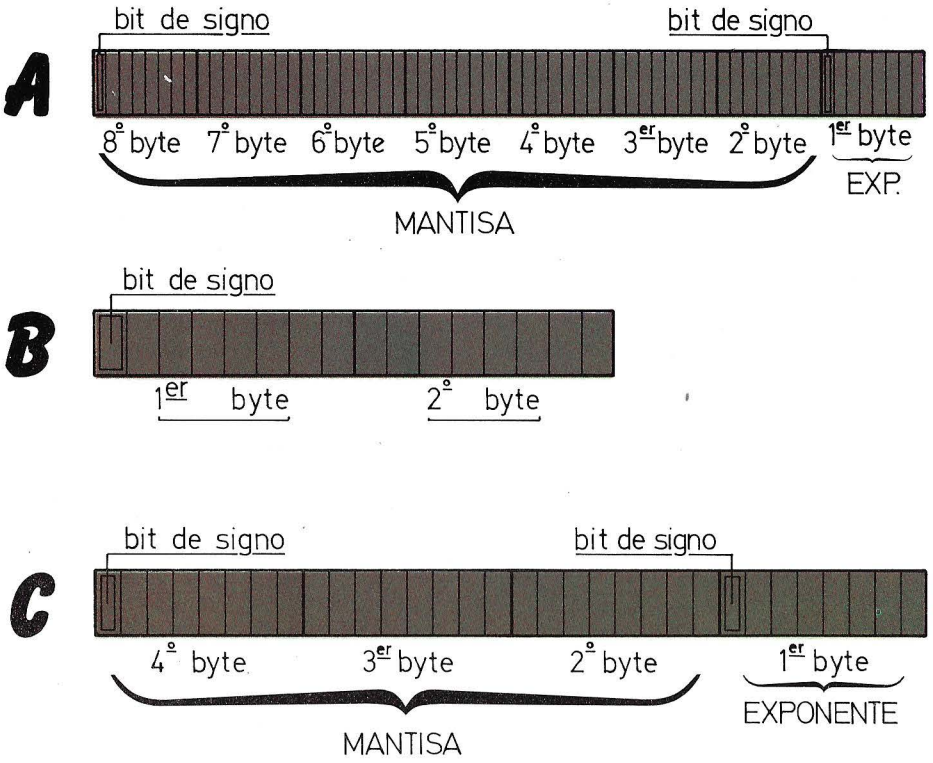
- Más de siete dígitos de precisión.
- Forma exponencial con D.
- Terminada con el signo de número ( ).

Los números de precisión sencilla se almacenan con siete dígitos de precisión, mostrándose únicamente los seis más significativos, mientras que los de doble precisión se almacenan con dieciséis dígitos.

OTROS SISTEMAS DE NUMERACION

Desde luego, es posible trabajar con otros sistemas de numeración distintos del de-

Estructura de los tipos básicos de representación numérica: doble precisión (A), números enteros (B), y simple precisión (C).



cimal (de base 10). Sin ir más lejos, el ordenador utiliza su propio sistema de numeración: el binario o de base 2. Ello se debe a la facilidad que tiene la máquina para manejar "ceros" y "unos" (los únicos dígitos del sistema binario).

Lo que resulta muy cómodo para el ordenador se vuelve claramente incómodo para el usuario. Así, por ejemplo, el número 8 decimal, se expresa en binario como 1000; análogamente, el número decimal 1000 necesita nueve cifras en su expresión binaria. El problema se agrava por el hecho de que no es inmediata la conversión de binario a decimal, y viceversa.

Los sistemas de numeración más fácilmente transportables al binario son aquellos cuya base es una potencia de 2 (base 4, 8, 16, 32, ...). Entre ellos, los más cercanos a la base 10, y por ende los más

Representación de números negativos

Los ordenadores operan y almacenan los números de acuerdo al sistema de numeración binario. De esta forma, un número se convierte en una serie de unos y ceros, que son más fácilmente manipulables por un circuito electrónico. Este sólo ha de detectar la presencia o ausencia de tensión para así determinar cuál es el valor del dato procesado.

La suma de números en el sistema binario no ofrece dificultad alguna, puesto que se reduce a la suma de unos y ceros. En cambio, la resta resulta ser más complicada, sobre todo por la posibilidad de obtener un dato resultante negativo.

La representación de datos negativos en el ordenador, suele solventarse utilizando el bit situado más a la izquierda de la palabra binaria representativa del número en cuestión. Este se emplea como "bit de signo". Si el bit en cuestión se encuentra a "1", el número es negativo; por el contrario, si su valor es "0", el número será positivo (o viceversa, dependiendo del criterio elegido).

La magnitud del número en cuestión viene determinada por los restantes bits. Estos pueden representar simplemente el valor absoluto del número.

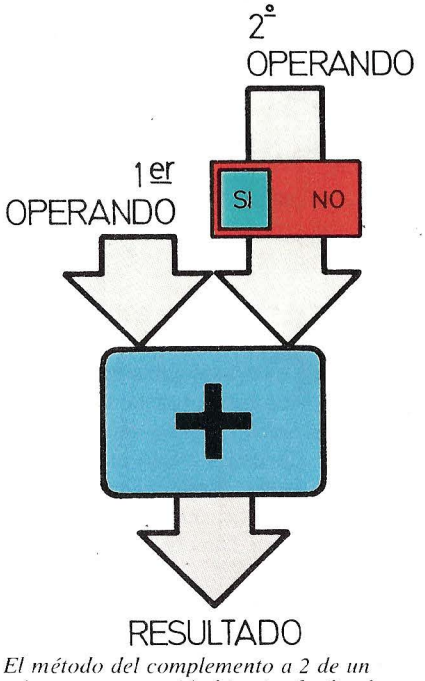
A pesar de todo, este modo de actuación presenta ciertos problemas a la hora de operar. Si se desea realizar la suma de dos datos positivos, el resultado será el correcto. Sin embargo, no ocurre así cuando uno de ambos números es negativo: el resultado coincidirá con un número negativo, de valor igual a la suma de los valores absolutos de los operandos.

Tal inconveniente se soluciona con un curioso método de codificación: el denominado "complemento a dos" del número positivo. Este método deriva del complemento a uno habitual, consistente en cambiar unos por ceros y viceversa. Así, por ejemplo, el complemento a uno del número 00100011 es 11011100.

Concretamente, el complemento a dos resulta de sumar una unidad al complemento a uno del dato base. Por lo tanto, el complemento a dos del número anterior será 11011101.

Como se observa, el bit de signo ha cambiado, revelando la presencia de un dato negativo. Se puede demostrar que al sumar números de distinto signo, representados de acuerdo a este método, el resultado obtenido coincide con la resta de los operandos.

Esta sencilla técnica permite reducir cualquier sustracción a una operación de suma, sin más que hallar previamente el complemento a dos del sustraendo.



El método del complemento a 2 de un número en expresión binaria, facilita la operación de sumar números de distinto signo o, lo que es lo mismo, la obtención de restas.

utilizados, son el octal (base 8) y el hexadecimal (base 16).
En BASIC se pueden especificar constantes octales y hexadecimales, e incluso operar con ellas. A fin de cuentas, en el interior de la máquina todo se convierte a binario.
—*Constantes hexadecimales:* son números expresados en el sistema de numeración hexadecimal o de base 16. Para su identificación han de ir precedidos por el prefijo &H.

Ejemplos: &H1A45, &H12F

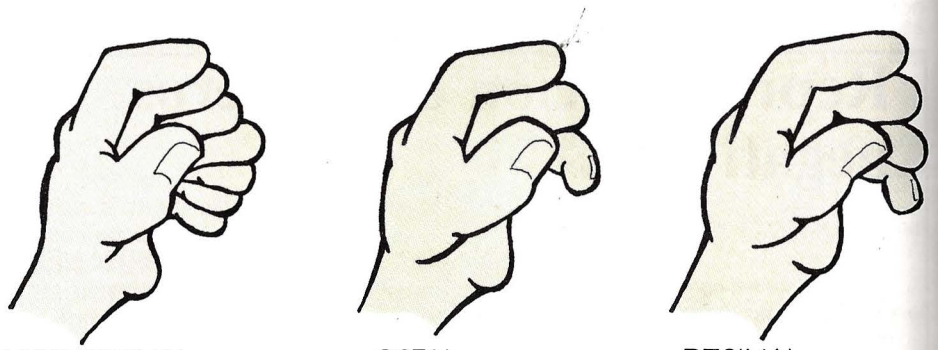
—*Constantes octales:* se trata de números expresados en base ocho (sistema octal). Se distinguen al ir precedidos por &O.

Ejemplo: &O0123, &O264

Una de las facultades básicas del ordenador reside en su aptitud para realizar cálculos numéricos. Los datos a operar son suministrados por el propio programa o bien son recogidos del exterior a través de los oportunos dispositivos periféricos.

VARIABLES

Como ya se ha explicado en capítulos anteriores, una variable puede equipararse a un cajón capaz de contener un dato. Realmente, la variable no es más que un espacio de memoria accesible por medio de un nombre característico: el nombre de la variable. Dicho espacio de memoria puede ver alterado su contenido durante la ejecución del programa. El hecho de que la variable identifique a una zona de memoria específica, implica que deben existir tantos tipos de variables como tipos de datos existan.
De cara al usuario, una variable es un nombre bajo el cual se guarda un contenido. Pero ¿qué restricciones hay en el uso de nombres de variable? Las respuestas difieren según el dialecto BASIC que utilice cada equipo.
La primera limitación está en el número de caracteres permitidos. La mayoría de los ordenadores admiten un número suficiente de caracteres como para identificar con comodidad el contenido de la variable. Así, los nombres VELOCIDAD o GASTOS serán casi siempre válidos. No obstante, otros aparatos sólo reconocen los primeros caracteres, ignorando los restantes. Para un ordenador de este tipo, capaz de reconocer tan sólo los dos primeros caracteres, los siguientes nombres harían referencia a la misma variable:



Además del decimal, hay otros sistemas de numeración perfectamente utilizables para el trabajo práctico. Los dos más difundidos son el octal (base B) y hexadecimal (base 16).

ALCANCE,ALETAS,AL,ALA,ALTURA

En tal caso, es aconsejable tomar buena nota de los nombres utilizados para no dar a dos variables un mismo identificador.
El nombre de una variable puede incluir, además de letras, números e incluso símbolos y signos de puntuación. Sin embargo, el primer carácter ha de coincidir siempre con una letra.
Un nombre de variable no debe coincidir con ninguna de las palabras reservadas del lenguaje BASIC (comandos o funciones). Si así fuera, el aparato no podría distinguir entre unos y otras. Existen ordenadores que no permiten tan siquiera que una palabra reservada forme parte de un nombre. En ellos no serían válidos los siguientes nombres o variables:

AN*FOR*A, DI*F*ERENCIA, *LET*RAS

TIPOS DE VARIABLES

Una variable puede almacenar un dato de tipo numérico o de tipo alfanumérico (cadena de caracteres). En el segundo caso,

el nombre de la variable debe terminar indefectiblemente con el símbolo de dólar ($).

Dentro de las variables de tipo numérico habrá que distinguir el formato de almacenamiento que utilizan.

En el propio nombre de la variable se puede especificar el formato deseado. Para ello sólo es necesario añadir un sím-

Del microprocesador al microordenador

En cualquier disciplina del saber humano, para conocer a fondo un sistema es necesario saber cómo está constituido. Para poder manejarlo con soltura hace falta tener presente su organización.
Un microordenador es un sistema basado en microprocesador. Sin lugar a dudas, el conocimiento de su estructura interna permite entender mejor su funcionamiento y adquirir una idea clara de sus posibilidades.
Cualquier ordenador consta de un conjunto de bloques fundamentales. Estos son la unidad central de proceso, el reloj, la memoria y la unidad de entrada/salida. A estas unidades internas del ordenador es necesario añadir otros dispositivos externos: los denominados periféricos. Cada una de estas zonas constitutivas del ordenador tiene su cometido específico. Todas ellas se comunican entre sí por medio de conductores de enlace que canalizan la información en forma de señales eléctricas. Estos cables pueden transmitir distintos tipos de información: datos y órdenes. Cada mazo o agrupación de conductores que permiten el intercambio de información entre los elementos del sistema se denomina BUS. La función de cada unidad básica de un sistema microprocesador es la que se indica a continuación.

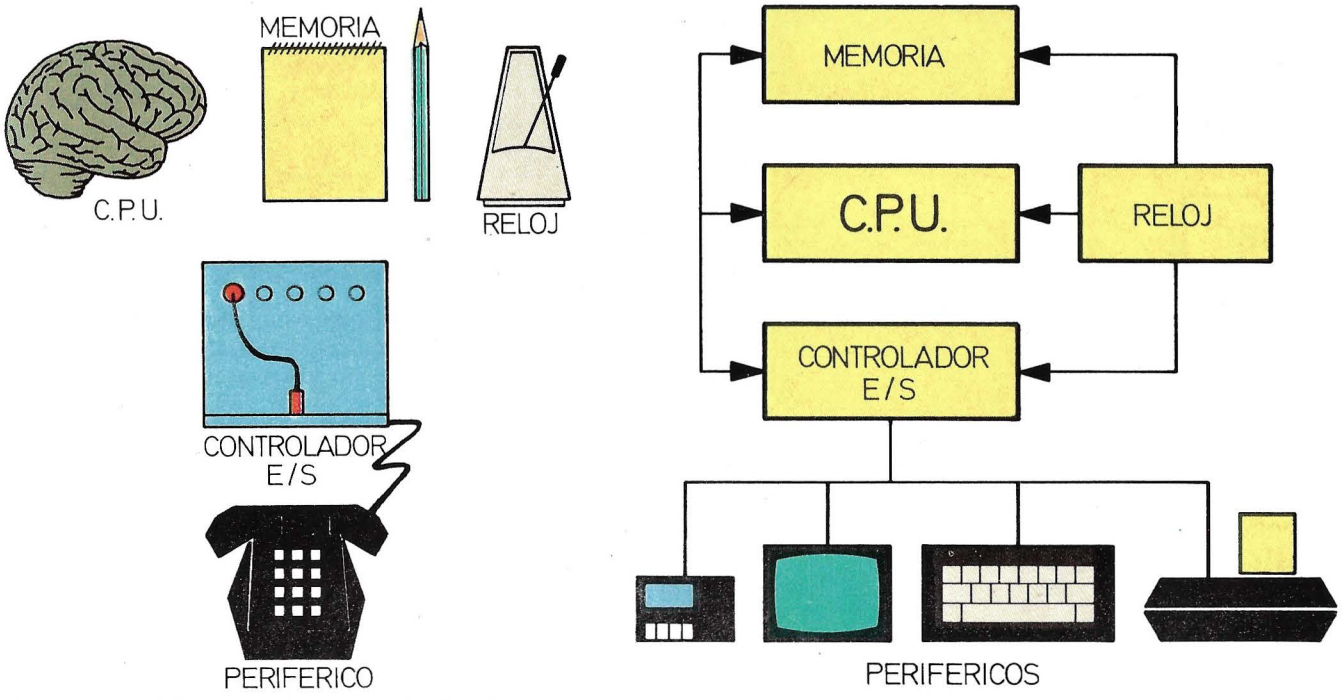
Unidad central de proceso (CPU): es el centro neurálgico del microordenador y está constituida por el circuito integrado microprocesador. Se encarga de realizar las operaciones aritméticas y lógicas y de interpretar las instrucciones del programa. La CPU es el auténtico cerebro del sistema y controla el funcionamiento de los restantes elementos y unidades que lo constituyen.

Reloj: el reloj es al ordenador lo que el director a la orquesta. Es precisamente el que marca el ritmo de operación. Gracias a este circuito, el conjunto de elementos asociados trabajan en armonía y sincronizadamente. La frecuencia de reloj de un ordenador da una idea de la velocidad de operación del mismo.

Memoria: está integrada por una serie de dispositivos electrónicos capaces de almacenar información binaria. La memoria es el bloc de notas del ordenador. En ella anota los datos y resultados de sus cálculos, así como la lista de tareas a realizar (el programa). Por regla general, cuanto mayor sea la memoria más potente será el ordenador. Esta magnitud se mide en Kbytes (1Kbyte=1024 bytes u octetos de información).

Unidad de entrada/salida: su función es la de facilitar la comunicación del ordenador con los dispositivos externos. Se encargan de adecuar los formatos y velocidades de la información a intercambiar entre el ordenador y los dispositivos periféricos.

Periféricos: son los dispositivos físicos que permiten la comunicación con el exterior. Sin ellos el ordenador estaría aislado y sería incapaz de recibir y emitir información. En los microordenadores los dos periféricos más importantes son el teclado y la pantalla.



Los elementos básicos que dan cuerpo al ordenador tienen un gran paralelismo con elementos asociados a la actividad humana: el cerebro o CPU (el microprocesador en los microordenadores), la memoria, el reloj que sincroniza el ritmo de trabajo y el controlador de E/S, que facilita el intercambio de información con el mundo exterior a través de los dispositivos periféricos.

bolo al final, de la misma forma que para las alfanuméricas. Estos símbolos especiales son los siguientes:

% Para variables de tipo entero.
! Para variables de simple precisión.
Para las de doble precisión.

Como se puede observar, estos identificadores de tipo se corresponden con los utilizados para las constantes.

Cuando no se especifica el formato, el ordenador entenderá que se trata de una variable de simple precisión.

DEFINICION DE TIPOS DE DATOS

Mediante determinadas funciones es posible definir el tipo de dato que almacenarán distintas variables. Sin embargo, en el caso de utilizar ambos métodos de definición, prevalecerá el impuesto por los caracteres indicados en el apartado precedente.

DEFINT

Define las variables cuya inicial se especifica en el formato de números enteros.

FORMATO: DEFINT <inicial> [,<ini1>–<ini2>]...
EJEMPLOS:

```
10 DEFINT H , A–C
50 DEFINT I , L , Z, R
```

DEFSNG

Define las variables cuya inicial se especifica en el formato de números de simple precisión.

FORMATO: DEFSNG <inicial> [,<ini1>–<ini2>]...
EJEMPLOS:

```
10 DEFSNG X–Z, B
50 DEFSNG E , L , P
```

DEFDBL

Define las variables cuya inicial se especifica en el formato de números de doble precisión.

FORMATO: DEFDBL <inicial> [,<ini1>–<ini2>]...
EJEMPLOS:

```
10 DEFDBL K , C–G
30 DEFDBL H , E , J
```

MOD

Calcula el valor del resto asociado a una división entera.

FORMATO: <dividendo> MOD <divisor>
EJEMPLOS:

```
20 LET TGN=17 MOD 3
70 PRINT A MOD 2
```

Las funciones de definición permiten elegir el formato de representación interna de una o varias variables. Dicha definición se realiza indicando la inicial de la (o las) variable(s) que utilizarán dicho formato. Las funciones encargadas de este cometido son las siguientes:

DEFINT: Define a las variables correspondientes enteras.

Su formato adopta el siguiente aspecto:

(Número de línea) DEFINT <letra>, [<letra de comienzo> – <letra final>] ...

Mediante esta función se declara que todas las variables que comiencen por la letra indicada serán de tipo entero. Asimismo, también pertenecerán a ese tipo aquellas variables cuya inicial se encuentre entre las letras de comienzo y final. Se pueden especificar tantas definiciones como sea necesario bajo un mismo enunciado; por ejemplo:

```
10 DEFINT I,K–M
```

La definición dada como ejemplo fuerza a que toda variable que comience por las letras I,K,L y M se mantenga en el formato entero.

DEFSNG: Las variables afectadas serán tomadas por el ordenador como variables de simple precisión.

Se rige por el siguiente formato:

(Número de línea) DEFSNG <letra>, [<letra comienzo> – <letra final>]...

El uso de esta función es idéntico al de la anterior, sin más diferencia que el tipo de formato que especifica. Por ejemplo:

```
20 DEFSNG A – F,H
```

En este ejemplo, las variables cuya inicial sea A,B,C,D,E,F o H pertenecerán al tipo de simple precisión.

DEFDBL: Declara a un conjunto de variables como de doble precisión.

Su formato es similar al de las anteriores:

(Número de línea) DEFDBL <letra>, [<letra de comienzo> – <letra final>]...

TABLA DE CONVERSION

ORDENADOR	DEFINICION DE TIPO			DIVISION ENTERA	RESTO
	DEFINT	DEFSNG	DEFDBL	/	MOD
APPLE II (APPLESOFT)	—	—	—	—	—
APRICOT (M-BASIC)	DEFINT	DEFSNG	DEFDBL	/	MOD
ATARI	—	—	—	—	—
CBM 64	—	—	—	—	—
DRAGON	—	—	—	—	—
EQUIPOS MSX	DEFINT	DEFSNG	DEFDBL	/	MOD
HP-150	DEFINT	DEFSNG	DEFDBL	/	MOD
IBM PC	DEFINT	DEFSNG	DEFDBL	/	MOD
MPF	—	—	—	—	—
NCR DM-V (MS-BASIC)	DEFINT	DEFSNG	DEFDBL	/	MOD
NEW BRAIN	—	—	—	—	—
ORIC	—	—	—	—	—
SHARP MZ-700 (MZ-BASIC)	—	—	—	—	—
SINCRAIR QL	—	—	—	DIV	MOD
SPECTRAVIDEO	DEFINT	DEFSNG	DEFDBL	/	MOD
ZX-SPECTRUM	—	—	—	—	—

El funcionamiento de DEFDBL es análogo al de las otras dos funciones:

30 DEFDBL Z,C – E

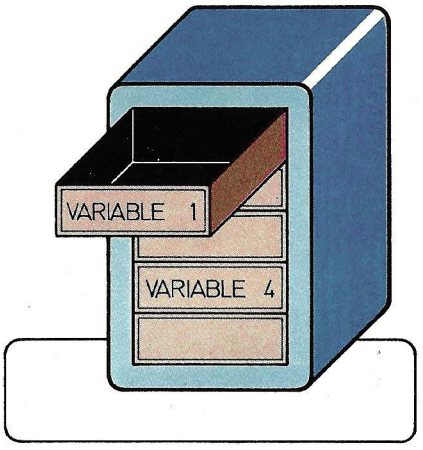
Ahora, las variables que empiecen por las letras C,D,E y Z serán almacenadas en formato de doble precisión.

OPERACIONES CON DISTINTOS TIPOS DE DATOS

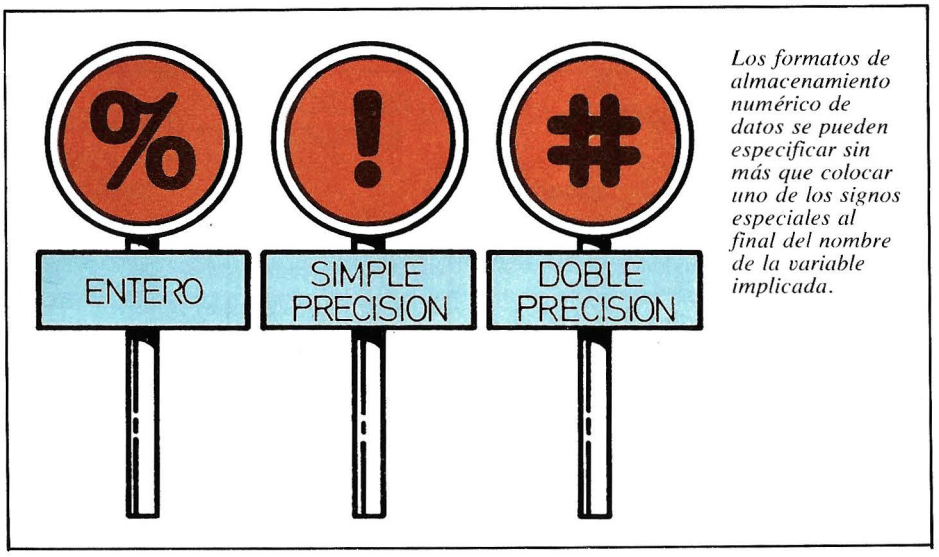
En ocasiones resulta necesario operar con variables y constantes de distintos tipos. En tal caso, el ordenador reconvierte el tipo de los operandos antes de realizar la respectiva operación. Si se trata de una sentencia de asignación, los datos quedarán convertidos al tipo de la variable receptora del resultado.

A la hora de evaluar una expresión, todos los operandos se convierten al mismo tipo: el del operando más preciso.

Los operadores lógicos convierten en enteros a sus operandos, generando un resultado entero. Dichos operandos deben tener un valor comprendido entre –32768 y 32767. El resultado de una operación de esta categoría será 0 (falso) ó 1 (verdadero).



Estrictamente, las variables sirven para designar a una zona de memoria reservada para acoger valores constantes de un tipo determinado.



Los formatos de almacenamiento numérico de datos se pueden especificar sin más que colocar uno de los signos especiales al final del nombre de la variable implicada.

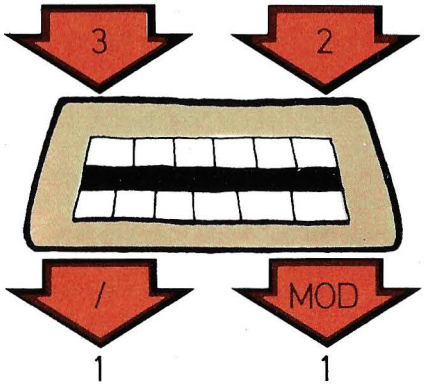
OPERANDO CON ENTEROS

Cuando se realizan operaciones con datos de tipo entero surge un problema: la división de dos números enteros no siempre da como resultado un número entero. Por ejemplo, si se desea hallar el cociente entre los números enteros 5 y 4 basta con escribir la siguiente línea:

```
PRINT 5% / 4% <CR>
1
■
```

El resultado obtenido es a todas luces erróneo. La razón de tan extraño comportamiento debe atribuirse a la naturaleza de los números enteros. Un número entero carece, por definición, de parte fraccionaria. Este hecho implica que el resultado ha de coincidir con el entero más próximo.
En todo caso, tal eventualidad está prevista en el lenguaje BASIC. La solución llega de la mano de un nuevo operador: el de la división entera. Esta operación permite hallar el cociente entero de la división entre dos números enteros. En otros términos: proporciona el entero más cercano inferior al cociente real.
El símbolo que identifica a esta operación es el denominado "backslash" ( ). Su diferencia respecto a la división normal se aprecia a la vista de los siguientes ejemplos:

```
PRINT 5 / 4 <CR>
1.25
PRINT 4 / 5 <CR>
.8
■
```



Además de la división convencional, el BASIC dispone de dos funciones adecuadas para operar divisiones con números enteros: el operador de división entera ("\" y la función MOD, capaz de obtener el resto entero de un cociente.

El ordenador ha tomado los datos en formato de simple precisión. Los resultados son números reales, puesto que se ha utilizado el operador de división convencional. Observemos qué ocurre cuando se utiliza la división entera.

```
PRINT 5% \ 4% <CR>
1
PRINT 4% \ 5% <CR>
0
■
```

Como era de esperar, los resultados son del tipo entero. Hasta aquí no hay nada sorprendente. En realidad, esta operación parece no ser más que la parte entera del resultado real: ahí va la demostración:

```
PRINT INT(5 / 4) <CR>
1
PRINT INT(4 / 5) <CR>
0
■
```

Sin embargo, la división entera guarda aún una sorpresa. Por medio de ella es posible determinar también el resto de la división. Como quiera que el resto se denomina en inglés "module" (módulo), la función asociada a la anterior y que proporciona el valor del resto recibe el nombre de MOD:

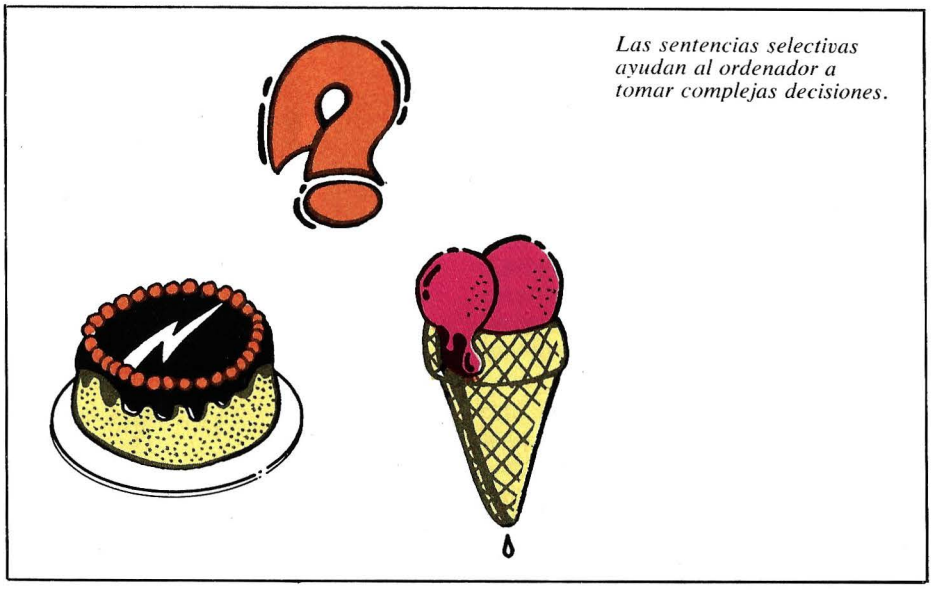
```
PRINT 5% \ 4% <CR>
1
PRINT 5% MOD 4% <CR>
1
PRINT 4% \ 5% <CR>
0
PRINT 4% MOD 5% <CR>
4
```

Los ejemplos reflejados ilustran el uso combinado de ambos operadores: división entera y módulo de un cociente.

Pascal (5)

Estructuras selectivas

Para concluir con el estudio de las estructuras fundamentales de la programación estructurada, hay que examinar las denominadas estructuras selectivas o condicionales. Su especialidad es la de romper el flujo del programa en un determinado punto del mismo, dependiendo de si se cumple o no una condición establecida. El valor lógico de la condición se comprueba durante el desarrollo del programa. Si es cierto (valor lógico TRUE) se ejecutará una sentencia específica, mientras que si se obtiene el valor lógico falso (FALSE), será otra la sentencia a ejecutar a continuación.



Las sentencias selectivas ayudan al ordenador a tomar complejas decisiones.

LA SENTENCIA IF/THEN/ELSE

La estructura selectiva está fielmente reflejada en la sentencia de control IF/THEN/ELSE. Esta adopta en el PASCAL un formato más elegante y claro que en otros lenguajes, como es el caso del FORTRAN o el BASIC. En efecto, ahora no es necesario utilizar números de línea a los que dirigir la ejecución del programa, sino que es posible introducir hasta programas completos como alternativa a cada una de las decisiones derivadas de la condición impuesta. Su formato general es:

IF <expr.> THEN <sent.> [ELSE <sent.>] ;

En ella, <expr.> equivale a una expresión de tipo booleano, cuyo resultado puede adoptar los valores TRUE o FALSE. A su vez, <sent.> corresponde a sentencias simples o incluso compuestas en cuyo caso deben ir agrupadas dentro de bloques BEGIN/END.

La zona ELSE puede ser omitida; en tal situación, se pasará a ejecutar directamente la sentencia posterior a la asociada a la zona THEN cuando no se verifique la condición impuesta.

Al tropezar con una sentencia de esta categoría, es fácil imaginar cuál será la actuación del ordenador. Si la condición es cierta, pasará a ejecutar la sentencia correspondiente a la zona THEN, ignorando la asociada a la parte ELSE; por el contrario, si la condición es FALSE, emprenderá la acción inversa. Así, por ejemplo, en el siguiente segmento de programa:

```
IF cierto
      THEN WRITELN('VERDADERO')
      ELSE WRITELN('FALSO');
```

"cierto" será una variable declarada previamente como de tipo booleano cuyo valor actual será visualizado en la pantalla. Una precisión relativa a esta sentencia, es que no hay que colocar un punto y coma ";" delante de la palabra ELSE, aun en el caso de que cada zona de la sentencia IF se escriba en líneas separadas.
Las sentencias IF pueden ser "anidadas", o más propiamente dicho, encadenadas una tras otra. Desde luego, al poner en práctica esta posibilidad habrá que tener cuidado para no perder de vista qué zonas THEN y ELSE corresponden a cada sentencia IF. En el siguiente ejemplo, la zona ELSE corresponde a la segunda sentencia IF, por ser la más cercana; aunque, obviamente, esta construcción puede resultar ambigua en ciertos casos:

```
IF cont>10   THEN   IF   suma=0
THEN   bal :=TRUE   ELSE
bal :=FALSE;
```

Para eliminar tal ambigüedad, se puede proceder de dos maneras. Una posibilidad es utilizar la indentación, admitida por el formato libre; ello supone alinear en la misma columna las zonas THEN y ELSE correspondientes a una misma sentencia IF. La segunda alternativa es englobarlas dentro de bloques BEGIN/END de la siguiente forma:

```
IF cont>10 THEN BEGIN IF suma=0
THEN bal :=TRUE ELSE
bal :=FALSE END;
```

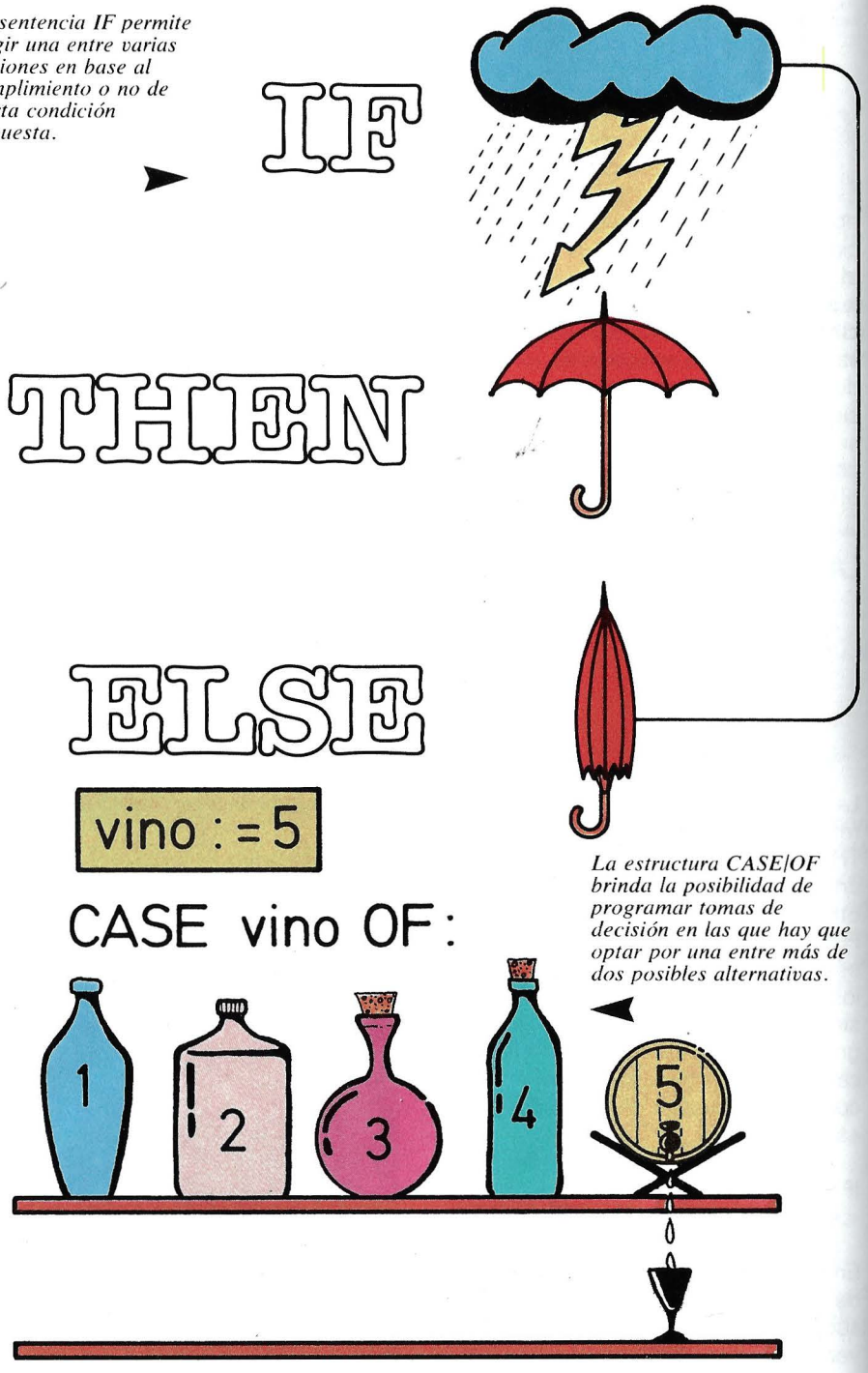
Cabe también la posibilidad de utilizar la siguiente formulación:

```
IF cont>10 THEN BEGIN IF suma=0
THEN bal :=TRUE END ELSE
bal :=FALSE;
```

si lo que se desea es que la zona ELSE, corresponda a la primera sentencia IF. Desde luego, con este método queda más claro a qué sentencias pertenece cada ELSE.
La aplicación más frecuente de la sentencia IF/THEN/ELSE es la de seleccionar una entre varias opciones. Veamos cómo se realiza en el siguiente ejemplo:

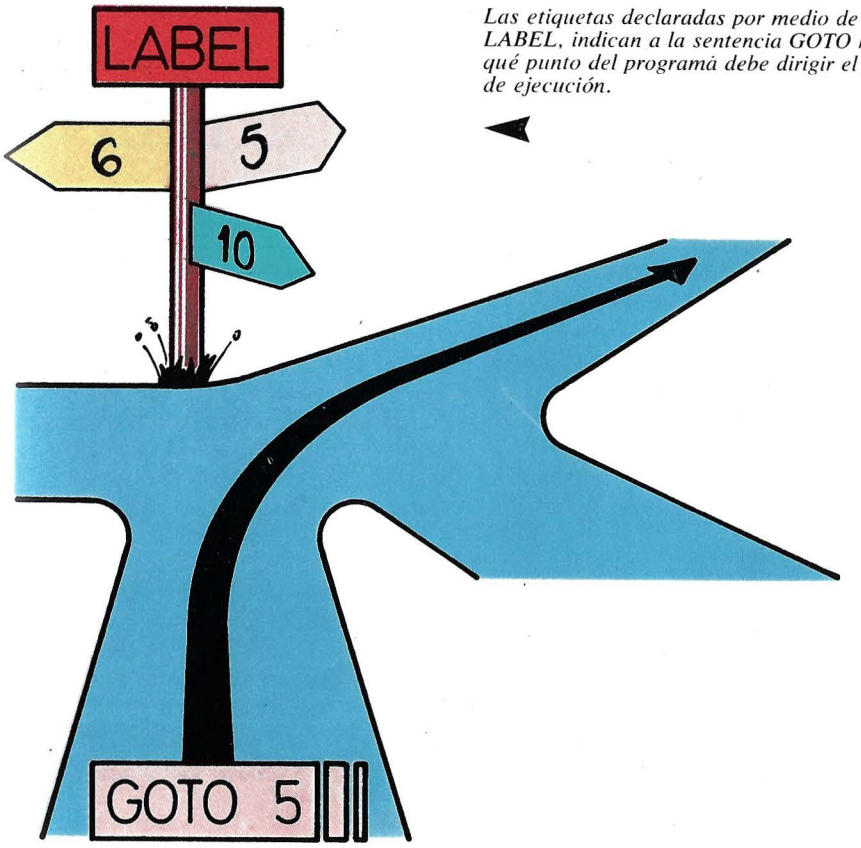
```
IF numero>9999 THEN
      cifras:=5
ELSE IF numero>999 THEN
      cifras:=4
ELSE IF numero>99 THEN
      cifras:=3
ELSE IF numero>9 THEN
      cifras:=2
ELSE cifras:=1;
```

La sentencia IF permite elegir una entre varias opciones en base al cumplimiento o no de cierta condición impuesta.

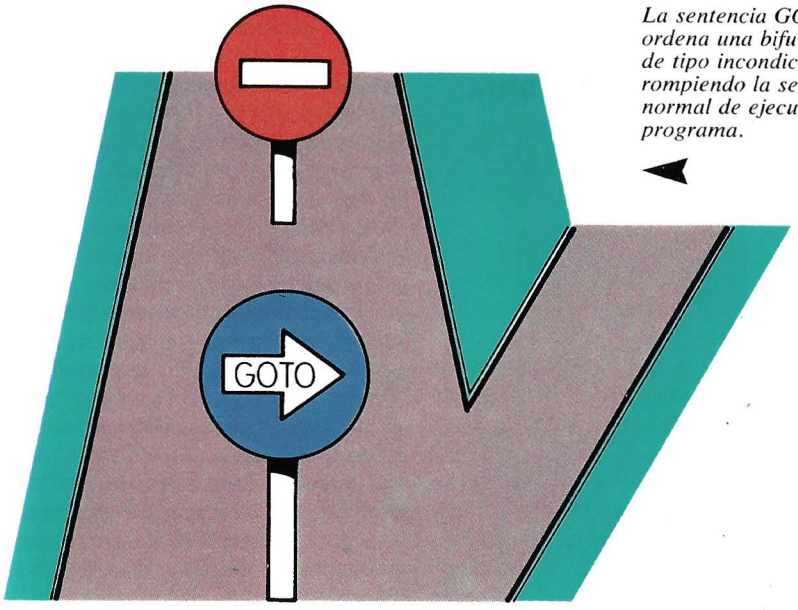


La estructura CASE/OF brinda la posibilidad de programar tomas de decisión en las que hay que optar por una entre más de dos posibles alternativas.

Esta construcción permite ejecutar la primera sentencia IF cuya condición sea cierta, abandonándose tras ello toda la estructura selectiva, de tal forma que sólo se ejecutará una de las sentencias de asignación incluidas en la misma.



Las etiquetas declaradas por medio de LABEL, indican a la sentencia GOTO hacia qué punto del programa debe dirigir el flujo de ejecución.



La sentencia GOTO ordena una bifurcación de tipo incondicional, rompiendo la secuencia normal de ejecución del programa.

MUESTRARIO PARA ELEGIR

La sentencia IF sólo permite elegir una entre dos opciones: ejecutar la zona THEN o ejecutar la zona ELSE. Sin embargo, en muchos casos interesa poder elegir una entre varias opciones (más de dos). Para que ello sea posible utilizando sólo sentencias IF, será preciso construir estructuras artificiales, similares a la del último ejemplo.

Sin lugar a dudas ésta es una posibilidad de indudable eficacia; pero queda aún otra sentencia de control, de tipo selectivo o de bifurcación condicional, que resolverá el problema con una mayor elegancia. Se trata de la sentencia CASE/OF, cuyo formato es el que se indica a continuación:

CASE <expr.> OF <et.>:<sent.>;
<et.>:<sent.>;... END;

Esta incluye una expresión, <expr.>, de tipo ordinal (INTEGER,CHAR,...), que seleccionará una única sentencia, simple o compuesta, de todas las emplazadas a continuación de la palabra OF y hasta la palabra END, que pone punto final a la sentencia CASE.

Cada una de las sentencias que forman el muestrario de opciones alternativas, debe ir precedida por una constante del mismo tipo que el de la expresión que actúa como selector. Estas constantes son denominadas "etiquetas CASE" (una de las pocas cosas que no hay que declarar directamente en un programa PASCAL).

La estructura CASE seleccionará, en definitiva, aquella sentencia cuya etiqueta coincida con el valor actual que tomé la expresión selectora, ignorando a las restantes. Una vez ejecutada la sentencia elegida, el flujo del programa abandonará la estructura CASE/OF.

Como es habitual, una etiqueta puede preceder a todo un conjunto de sentencias englobadas dentro de un bloque BEGIN/END. Si no existe una etiqueta coincidente con el valor obtenido por la expresión selectora, el efecto de la sentencia CASE es indefinido, produciendo normalmente un mensaje de error en la pantalla. Por ejemplo:

TABLA DE COMANDOS-PASCAL		
COMANDO	COMETIDO	OBSERVACIONES
IF <expr.> THEN <sent.> { ELSE <sent.>};	Ejecución condicional de una entre dos opciones.	Cuerpo del programa.
CASE <sel.> OF <et.>:<sent.>;...END;	Ejecución condicional de una entre dos o más opciones.	Cuerpo del programa.
GOTO <etiqueta>;	Bifurcación incondicional.	Cuerpo del programa.
LABEL <etiqueta>;	Declaración de etiquetas.	Zona de definiciones y declaraciones.

NOTA: <expr.>: expresión de tipo booleano. <sent.>: sentencia simple o compuesta. <sel.>: expresión de tipo ordinal. <et.>: etiqueta que será una constante del mismo tipo que el selector. <etiqueta>: entero de no más de 4 cifras y sin signo.

El uso de GOTO resulta desaconsejable en PASCAL, puesto que rompe la estructura lógica del programa.

```
READLN(dia);
CASE dia OF
      1:nombre:='LUNES';
      2:nombre:='MARTES';
      3:nombre:='MIERCOLES';
      4:nombre:='JUEVES';
      5:nombre:='VIERNES';
      6:nombre:='DESCANSAR';

END;
WRITELN(nombre);
```

Como se ha indicado, si en el ejemplo anterior el valor de "dia" fuese menor que 1 ó mayor que 7, el resultado sería indefinido. Para evitar este tipo de errores es conveniente limitar de alguna forma los valores que pueda tomar la expresión selectora. Ello puede lograrse de dos formas. La primera es incluir la sentencia CASE en una de las zonas de una sentencia IF:

```
IF dia>0 AND dia<8 THEN
      CASE dia OF...
ELSE WRITELN('CODIGO ERRONEO');
```

De esta forma, si "dia" toma un valor no comprendido entre 1 y 7, ambos incluidos, se imprimirá el mensaje que indica el error.
La segunda alternativa es declarar la variable selectora como de tipo subrango, con lo cual sólo podrá tomar determinados valores:

```
VAR dia: 1..7;
```

BIFURCACION INCONDICIONAL

Para finalizar con las sentencias de control del PASCAL, queda por ver la de salto incondicional GOTO. Por medio de esta sentencia se puede dirigir el flujo del programa a cualquier punto del mismo, anterior o posterior a la posición que ocupe GOTO. Su formato general es muy simple; consta únicamente de la palabra clave GOTO, seguida por una etiqueta. Esta etiqueta, que será un número entero sin signo y, generalmente, de no más de cuatro cifras, se corresponderá con el mismo valor que se sitúe inmediatamente delante de la sentencia a la que se desea transferir la ejecución del programa.
Cada etiqueta que vaya a ser utilizada con una sentencia GOTO, debe ser declarada previamente en la zona de declaraciones y definiciones del programa, mediante la palabra clave LABEL; ésta precederá a cualquier declaración CONST o VAR.

```
PROGRAM ejemplogoto (OUTPUT);
LABEL 100,200;
VAR n:INTEGER;
BEGIN
      n :=1 ;
100:IF n>10 THEN GOTO 200;
      WRITELN(n);
      n :=n+1;
      GOTO 100;
      200:END.
```

En el programa ejemplo se producirán bifurcaciones a las sentencias "etiquetadas" con los números 100 y 200 en diversos momentos de la ejecución.
El uso de sentencias GOTO rompe la estructuración del programa y, en consecuencia, dificulta su seguimiento y suele ser causa de múltiples errores. De ahí que su presencia sea desaconsejable en muchos casos; aunque hay veces que resulta imprescindible recurrir a ella, ya que su eliminación podría complicar el programa en exceso. En todo caso, una máxima fundamental de la programación estructurada aboga porque todas las sentecias GOTO pueden ser eliminadas de un programa.
"La presencia de GOTO indica que el programador no ha aprendido a pensar en PASCAL» (Jensen & Wirth).

MS/DOS (7)

Gestión del almacenamiento en discos

El almacenamiento de datos para su conservación y uso posterior, supone el empleo de unidades de memoria externa. Entre éstas, las más ampliamente utilizadas en el mundo de la informática son las que utilizan discos magnéticos, flexibles o rígidos, como soporte de almacenamiento.

En el disco, la información está organizada en ficheros de diverso contenido. Algo semejante a lo que ocurre con las carpetas alojadas en un cajón de archivo, cada una de ellas destinada a acoger información de diversa índole.

LA DISTRIBUCION DEL ESPACIO DISPONIBLE

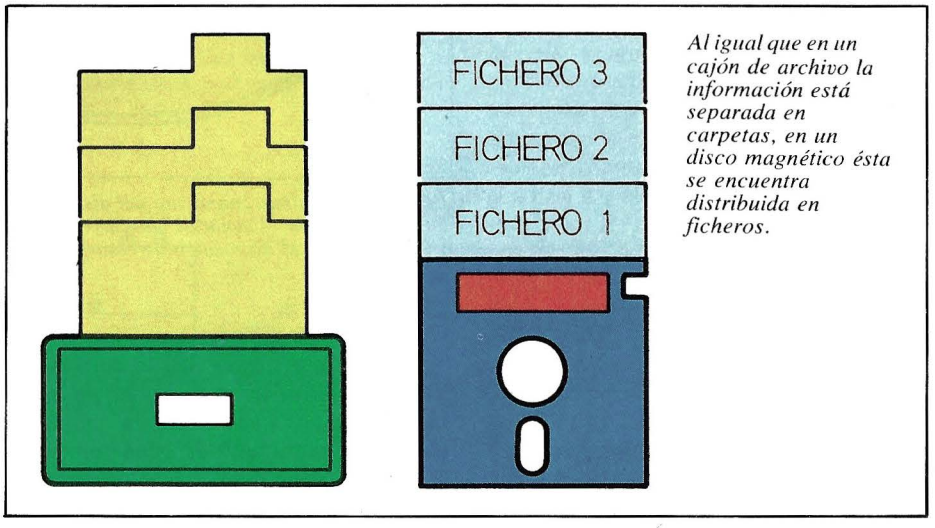
Todo dispositivo de almacenamiento tiene una capacidad nominal, expresada en Kilobytes (Kb) para los disquetes o discos flexibles y en Megabytes (Mb) en el caso de los discos rígidos. Esta medida indica la cantidad teórica de información que pueden contener. Sin embargo, no todo este espacio es accesible al usuario. El dispositivo que controla el disco o disquete precisa cierta información para su manejo; información que debe ser también almacenada y para lo cual reserva parte del espacio disponible.

Esto es lo que sucede cuando se formatea un disco virgen en orden a permitir su uso posterior. Si no se realizara el proceso de formateo o inicialización, el disco no sería reconocible por el sistema, al no contener ninguna información identificativa. La información residente en un disco o disquete se agrupa en bloques o conjuntos de bytes susceptibles de ser leídos o escritos con una sola operación de lectura o escritura. A este conjunto de bytes se le denomina *sector* y su tamaño varía según sea la densidad de almacenamiento que es capaz de soportar la superficie del disco. Son típicos, por ejemplo, los sectores de 512 bytes para discos flexibles de una cara y simple densidad. La siguiente etapa a la hora de construir la estructura de la información almacenada en un disco, es la agrupación de sectores hasta cubrir la longitud de un círculo completo. A esta agrupación de sectores se la conoce como *pista* o "track"; en la mayoría de los formatos existentes en el mercado, la pista está integrada por ocho sectores.

Si existe más de una superficie en el disco o disquete sobre la que es posible grabar la información (discos de doble cara), nace el concepto de *cilindro,* que incluye a la pista existente en la cara superior y a la localizada en la cara inferior. Al respecto, cabe dar la definición de cilindro como "conjunto de pistas que se encuentran a la misma distancia del centro".

Como consecuencia de su definición, los sectores situados en el mismo cilindro pueden ser leídos o escritos sin necesidad de desplazar el mecanismo de lectura/escritura; tan sólo hay que esperar a que el giro del disco ponga el sector afectado al alcance de la cabeza de lectura/escritura. En el caso de que para realizar la lectura o escritura fuese necesario el desplazamiento de la cabeza, el tiempo que tardaría ésta en posicionarse en el lugar correcto sería relativamente elevado; en consecuencia, la búsqueda sería lenta en comparación con la lectura/escritura en el mismo cilindro.

El número de pistas típico en un disquete suele ser de 40 a 80 por cada superficie activa del mismo. A su vez, los discos rígidos suelen estar organizados en unos 305 cilindros cada uno. Una vez que se han definido los diversos conjuntos de bytes en los que está organizado el disco, hay que examinar cuál es la disponibilidad del espacio así definido. Como se indicó al principio, no todo el espacio nominal puede ser utilizado por el usuario, sino que una porción del mismo queda reservado para el uso del sistema. En términos

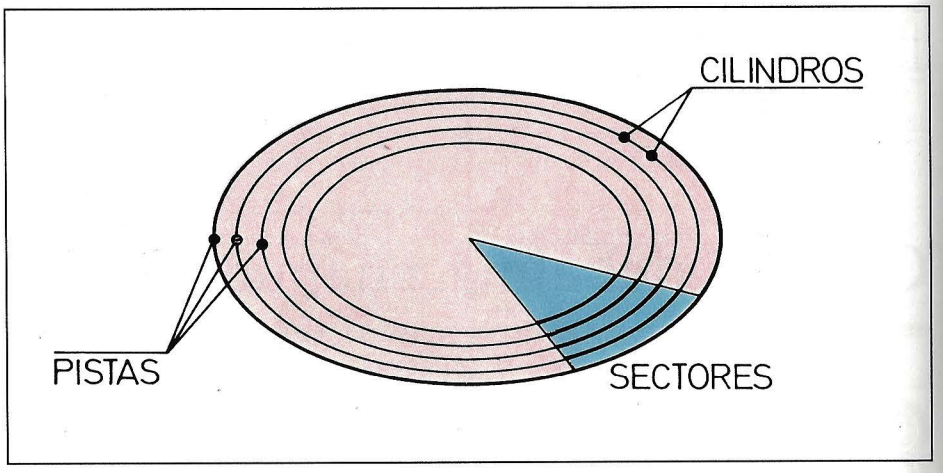


Al igual que en un cajón de archivo la información está separada en carpetas, en un disco magnético ésta se encuentra distribuida en ficheros.

generales, el espacio se distribuye de la siguiente forma:

— Registro de arranque o "boot record".
— Primera copia del archivo que contiene la tabla de asignaciones de espacio.
— Segunda copia del archivo que contiene la tabla de asignaciones.
— Directorio.
— Area de datos.

El registro de arranque contiene un programa, de reducida longitud, cuyo cometido es ayudar a la puesta en marcha del sistema operativo MS/DOS desde el disco. Este forma parte de todos los discos, aunque la mayor parte de ellos no albergan al sistema operativo sino tan sólo ficheros de datos.
Los espacios del disco utilizados están registrados en todo momento en el fichero que contiene la tabla de asignaciones de espacio o FAT (del inglés File Allocation Table). Dicha tabla memoriza dónde está localizado cada fichero y qué zonas del disco permanecen libre.
El directorio del disco es una tabla en la que reside una escueta información de cada uno de sus ficheros. Tal información incluye el nombre de cada fichero y el nombre de su extensión, el tipo del fichero, la fecha y hora en el que fue creado o modificado por última vez, su posición inicial dentro del disco y su longitud en bytes.

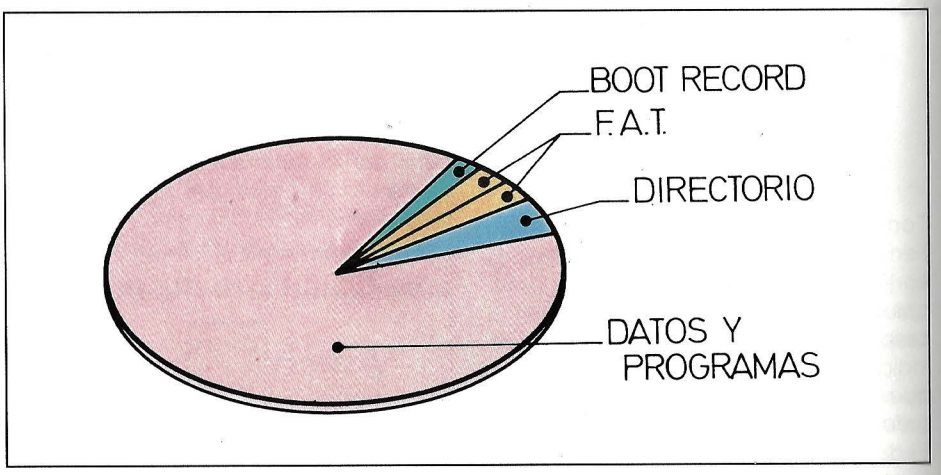


La distribución física del disco magnético se realiza en sectores; éstos, a su vez, se agrupan en pistas, componiendo estas últimas los cilindros.

DISCOS FLEXIBLES

Este medio de almacenamiento es el más frecuentemente utilizado en el campo de los microordenadores debido, básicamente, a su coste asequible y a las cada vez más altas prestaciones. Si bien, empiezan ya a ser reemplazadas por nuevos dispositivos más avanzados, hasta ahora exclusivos de ordenadores mucho más potentes.
Los discos flexibles, disquetes, o "floppies" están fabricados en plástico flexible. Ello permite que éste sea un medio de almacenamiento poco exigente para su conservación, facilitando su transporte y almacenamiento.

El disquete está constituido por una funda protectora de forma cuadrada, de plástico o cartón rígido, en la que aparecen varias ranuras y agujeros. El agujero central tiene por función dejar al descubierto la zona del disquete más próxima al centro, para que el mecanismo de arrastre sea capaz de sujetar al disquete y hacerlo girar. Dicha zona no almacena información alguna, ya que podría ser dañada fácilmente.



El espacio de almacenamiento del disco no está enteramente disponible para los ficheros que desee almacenar el usuario. Una determinada zona del mismo queda reservada para uso del sistema y para el dispositivo de control de los discos.

La ranura existente debajo del agujero central permite que se produzcan a través de ella las diversas operaciones de lectura/escritura, a realizar por la cabeza al efecto. En último término cabe mencionar la ranura existente en uno de los lados, en forma de pequeño entrante cuadrangular, que permite inhibir la escritura en el disquete si dicha zona aparece cubierta.
Los disquetes más comunes son los de 5 y 1/4 pulgadas, aunque actualmente empiezan a proliferar los de 3 y 1/2 pulgadas. Desde luego, también se utilizan los de ocho pulgadas, sobre todo en equipos antiguos.
Otra característica a destacar es la densidad de la información grabada en el disco: simple, doble y cuádruple densidad con 24, 48 y 96 pistas por pulgada, respectivamente. Por supuesto, el disco puede estar grabado por una sola cara o ambas.
Respecto a la posibilidad de formateado de los disquetes, cabe señalar que puede realizarse bajo el control de un programa

al efecto, aunque también hay discos cuya distribución o formato queda en el proceso de fabricación. En el primer caso, la sectorización es de tipo "soft" (soft-sectored), mientras que en el segundo hay que hablar de discos "hard-sectored". Por último, la capacidad de almacenamiento tiene un amplio margen de variación: desde los 100 kbytes hasta algunos Mbytes.

DISCOS RIGIDOS

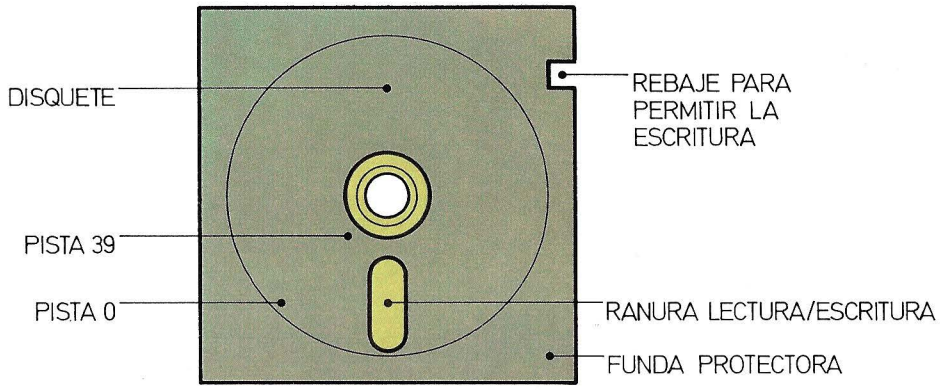
En los ordenadores que disponen de disco rígido, éste suele orientarse a las operaciones de almacenamiento en régimen de trabajo. En tal caso, los discos flexibles, dada su menor capacidad y velocidad de acceso, se reservan para la obtención de copias de seguridad del contenido de los discos rígidos.

Los discos rígidos están fabricados en base a un soporte rígido recubierto de material magnético. Una de las tecnologías más frecuentes es la denominada Winchester; de ella procede el uso extendido de llamar discos Winchester a la generalidad de discos rígidos.
Su capacidad de almacenamiento suele estar comprendida entre los 5 y los 50 Megabytes, con lo que sobrepasan ampliamente la capacidad de los disquetes. Los discos rígidos están sellados, debido a que la separación entre la superficie del disco y las cabezas lectoras es tan pequeña que cualquier partícula de polvo podría quedar atrapada en el espacio inter-

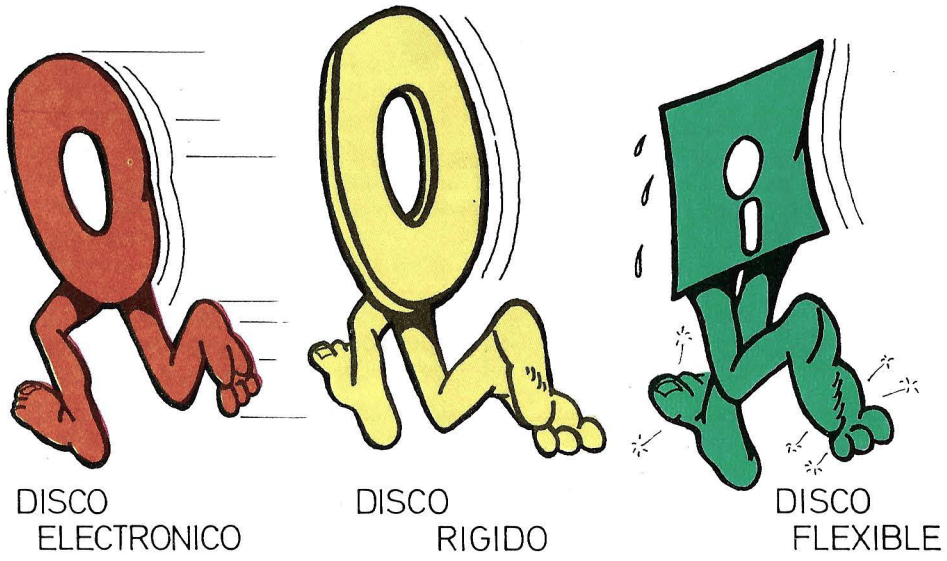
Programas y datos

El sistema operativo sólo es capaz de distinguir dos tipos básicos de ficheros entre la multiplicidad de ellos que pueden residir en un disco: los ficheros de *programas* y los de *datos*. En un ajuste más fino, cabe distinguir ciertas particularidades dentro de cada uno de ambos grupos. En cuanto a los ficheros cuyo contenido son programas ejecutables, el MS/DOS distingue entre dos categorías: aquellos cuya extensión es COM y los de extensión EXE.
Los ficheros de tipo COM se utilizan para almacenar programas que están listos para su ejecución inmediata; su formato refleja la imagen que éstos tienen en la memoria del ordenador cuando el programa está en curso de ejecución. A su vez, los ficheros de tipo EXE son programas que requieren un último retoque antes de ser ejecutados; retoque necesario para que el programa sea cargado en la zona de memoria adecuada. Este último formato se utiliza con programas complejos y se obtiene como resultado de la compilación de dichos programas.
Desde el punto de vista del ordenador sólo existen dos tipos de programas. Un programa BASIC se considera como un tipo especial de fichero de datos, ya que su contenido no es ejecutable directamente. En efecto, hay que utilizar para ello un programa intérprete que a la vista de los datos contenidos en el fichero BASIC realice la acción oportuna.
Dentro de los ficheros de datos, merecen una mención especial los que están grabados con formato ASCII. En ellos, los caracteres se almacenan de acuerdo al código alfanumérico ASCII. La gran ventaja que aporta este tipo de ficheros de datos reside en su notable universalidad. El uso del código ASCII permite el fácil transporte de los datos contenidos en un fichero de un sistema a otro. De ahí que su empleo sea recomendable siempre que se prevea que los datos residentes en un fichero puedan ser de interés en otros sistemas.

Bajo la perspectiva del sistema operativo, existen dos tipos básicos de ficheros: los de programas y los de datos.



El disco flexible consiste en una lámina circular, de material plástico, y recubierta de una capa de óxido magnetizable. El disco queda protegido por una funda cuadrada, de cartón, sobre la que aparecen diversas ranuras que dan acceso al disco interno.



La velocidad del disco electrónico o disco RAM es superior a la de los discos rígidos; ésta, a su vez, supera en más de cinco veces a la de los discos flexibles.

medio, pudiendo dañar la superficie magnética del disco.
Así pues, los discos no pueden ser extraídos fuera del mueble que los aloja, contrariamente a lo que ocurre con los discos flexibles. Pierden su movilidad, aunque su gran capacidad de almacenamiento hace que no sea necesaria su sustitución por otro disco vacío en la mayor parte de los casos. No obstante, existen discos rígidos contenidos en un cartucho que permite su extracción del dispositivo de control y la inserción en el mismo de un nuevo disco.
Debido a su especial y delicado dispositivo de lectura/escritura, la velocidad de las operaciones sobre un disco rígido supera en más de cinco veces a la habitual en un disquete. Por lo que respecta a la distribución del espacio, hay que señalar que a la hora de definir el disco mediante el comando FDISK, se pueden asignar hasta un máximo de cuatro zonas diferentes llamadas particiones, pudiendo ser cada una de ellas de distinto tamaño. El formateo de cada una de las particiones también puede diferir. De esta forma es posible conseguir que una partición contenga a un sistema operativo y sus ficheros correspondientes, y en otra aloje a un sistema operativo diferente. Esto es: se puede formatear una zona apropiada para el MS/DOS, e incluir en ella este sistema operativo, mientras que otra se puede inicializar con formato CP/M-86 y almacenar el sistema operativo CP/M-86.
Las particiones vienen a ser compartimentos estancos, de tal forma que no es posible transferir directamente datos de una partición a otra y, desde luego, tampoco puede estar un fichero a caballo entre dos particiones.

MICRODISCOS Y DISCOS ELECTRONICOS

Los microdiscos, en sus formatos de 3'' y 3 y 1/2 pulgadas —básicamente estos últimos—, están entrando con fuerza en el terreno de la microinformática. Su presencia es cada vez más común en equipos con MS/DOS. Los microdiscos aparecen encerrados en una caja de plástico rígido, en la que existe una trampilla que pone al descubierto una porción de superficie magnética. Otro tipo de disco, de reciente actualidad, es el llamado disco electrónico o disco RAM. Estrictamente, no se trata de un sistema para el almacenamiento de información, sino de una combinación de memoria y programas que permiten simular el funcionamiento de un disco en memoria RAM.
El motivo principal que aboga por la utilización de este tipo de dispositivos, reside en la gran diferencia de velocidad existente entre la CPU y las unidades de disco; ello hace que se produzca un cuello de botella en la velocidad de ejecución cuando un programa se ve obligado a acceder reiteradamente al disco. Con un disco electrónico se pueden mover datos desde un disco convencional hasta el disco electrónico, de manera que las operaciones de lectura y escritura de los datos durante el proceso de ejecución se realicen a mayor velocidad.
Debido a su naturaleza no se trata exactamente de un sistema para almacenar información, sino para tratarla. Al desconectar el suministro de corriente, los datos almacenados se pierden, siendo por tanto un sistema de memoria volátil. Por ello, el método de trabajo con el disco electrónico se concreta en los siguientes puntos:

— Los datos se transfieren desde un sistema de almacenamiento permanente (disco o disquete) al disco electrónico.
— Se realizan las operaciones oportunas con los datos en el disquete electrónico.
— Una vez terminado el proceso, los datos vuelven a ser memorizados en un disco o disquete.

Multitexto (1)

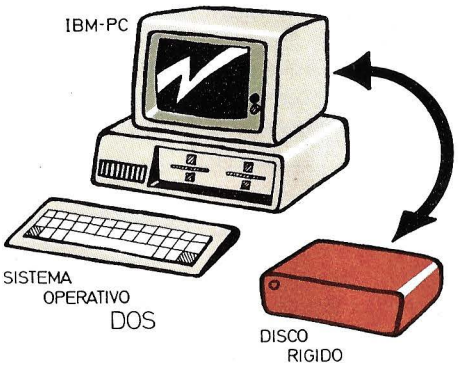
Características del procesador de textos

Aunque su origen no es español —se trata del procesador de textos MULTIMATE, nacido en Estados Unidos— éste es uno de los escasos paquetes de aplicación traducido al castellano. En nuestro idioma, su denominación original se ha sustituido por MULTITEXTO. Los derechos sobre este producto pertenecen a la compañía Multi Mate International Corporation y el distribuidor exclusivo para España es la firma APL Informática.
MULTITEXTO es un procesador de textos tradicional que aporta, entre otros detalles de interés, una novedad fundamental respecto a otras aplicaciones similares: la buena traducción al castellano tanto de los mensajes del programa como de la documentación. Sin duda alguna, ello facilita en gran medida su empleo por personas no anglófonas.

EL ENTORNO DE TRABAJO

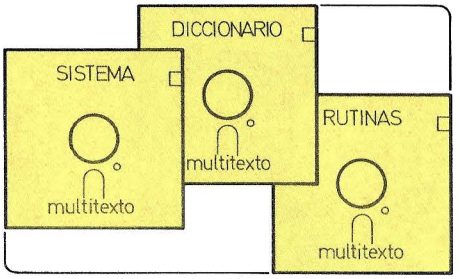
El paquete MULTITEXTO está concebido para operar con un ordenador personal IBM o compatible. El sistema operativo sobre el que se apoya es el MS/DOS en cualquiera de sus versiones. Exige una memoria principal disponible de al menos 256 Kbytes.
Junto con la aplicación, se suministra un manual de usuario dividido en dos volúmenes, redactado en castellano, y tres discos flexibles cuyo respectivo contenido es el siguiente:

1. Sistema central del procesador MULTITEXTO, en el que residen los programas fundamentales para el funcionamiento de la aplicación.
2. Rutinas auxiliares que sirven para que los programas del sistema central descarguen en ellas ciertas tareas.
3. Diccionario para la corrección de faltas ortográficas. Lamentablemente, el contenido de este disquete no está traducido, por lo que su uso queda restringido a textos en inglés.



La configuración del sistema adecuado para el uso de la aplicación MULTITEXTO, incluye al ordenador personal (IBM-PC o compatible) equipado con el sistema operativo MS/DOS (PC/DOS) y la unidad de almacenamiento (disco flexible, con o sin unidad adicional de disco rígido). Por supuesto, a ello hay que añadir la impresora adecuada para la obtención de copias en papel.

Los dos últimos discos se pueden copiar en un disco rígido; en cambio, el primero está especialmente diseñado para imposibilitar su copia. De esta forma, los propietarios de la aplicación se protegen de empresas de dudosa moralidad dedicadas a la reproducción y venta ilegal de aplicaciones informáticas. Como acabamos de apuntar, si se dispone de un disco rígido de gran capacidad, es deseable copiar los ficheros de la aplicación en este soporte. No obstante, si el ordenador donde se vaya a explorar la aplicación no dispone de disco rígido, habrán de utilizarse directamente los disquetes originales.
Para empezar el trabajo con el procesador MULTITEXTO, una vez incorporado el sistema operativo e introducido el disquete principal en la unidad apropiada, hay que teclear la clave WP y, tras ello, accionar la tecla RETURN. Inmediatamente aparecerá en la pantalla la ficha de presentación del paquete MULTITEXTO, con el siguiente mensaje en la zona inferior: "PULSE LA BARRA ESPACIADORA PARA CONTINUAR".
Obviamente, el siguiente paso consiste en accionar la referida barra espaciadora para que aparezca en la pantalla el menú principal. Este incluye nueve opciones que estudiaremos más adelante.



El paquete MULTITEXTO consta de tres discos flexibles: disco del sistema (con el procesador de textos), disco de rutinas de utilidad (programas de adaptación a distintos tipos de impresoras...) y diccionario en inglés para corrección ortográfica.

"FILOSOFIA" DE FUNCIONAMIENTO

Todos los procesadores de textos ofrecen prácticamente el mismo conjunto de funciones para escribir, almacenar e imprimir documentos. Lo que realmente permite diferenciarlos es la facilidad de uso y la rapidez de ejecución. En este sentido, MULTITEXTO brinda una serie de menús muy fáciles de utilizar que permiten al

usuario navegar por los grandes apartados (crear un documento, imprimirlo, etc.). No obstante, el menú no es obligatorio; por ejemplo, cuando el usuario se encuentra introduciendo un texto puede hacer desaparecer el menú de la pantalla, con lo que se ganará espacio.

En cuanto a la comunicación entre el programa y usuario, existen dos tipos de relaciones. Por un lado, el MULTITEXTO enviará mensajes advirtiendo de posibles errores, y por otro también podrán observarse apuntes sobre la tarea concreta que está resolviendo el programa en un instante dado.

Otra forma alternativa de comunicación consiste en recurrir a las pantallas de ayuda, a las que puede accederse sin necesidad de salir del documento. Los menús de ayuda definen las características de las funciones del programa y muestran las instrucciones adecuadas para que el usuario pueda llevar a cabo su tarea. De alguna forma, cabe afirmar que los menús de ayuda constituyen una documentación adicional e interactiva del MULTITEXTO.

El elemento unitario de este programa es la página, de tal forma que el usuario debe especificar sus características una sola vez y las páginas se terminarán imprimiendo tal y como aparecen en la pantalla, excepto cuando se modifique alguna característica durante la impresión. El tamaño de cada página está delimitado por 150 líneas como máximo. Este sistema basado en la página, además de resultar cómodo es útil como método de seguridad, ya que cada vez que termine la manipulación de una de ellas, el programa almacenará su contenido automáticamente en el soporte de amacenamiento permanente.

TERMINOLOGIA DEL MULTITEXTO

De nuevo, hay que mencionar que los términos utilizados en el ámbito del tratamiento de textos son prácticamente los mismos, sea cual fuere el paquete específico del que se trate. No obstante, la terminología y la forma en que el sistema "obedece" ante un término no siempre coincide en todos los programas. Por ello, vamos a definir a continuación los conceptos fundamentales en la terminología del MULTITEXTO:

- *Línea de identificación*

Primera línea de cada pantalla de un documento. Indica el nombre, el número de página y los números de línea y columna en donde se encuentra la posición activa (el cursor). Cuando se utiliza alguna función especial, como copiar, eliminar o mover, aparecerán los mensajes al efecto en la parte derecha de la línea de identificación.

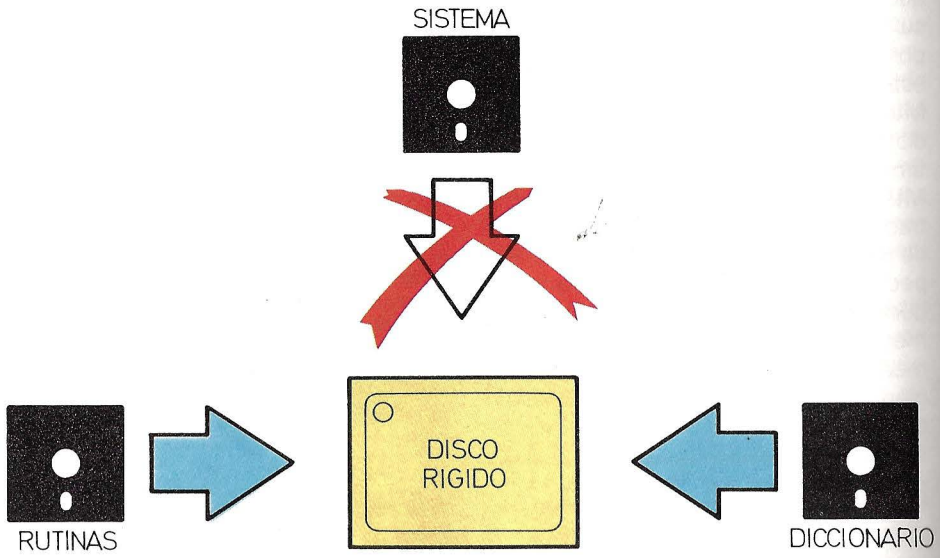
- *Línea de formato*

Cada página contiene al menos una línea de formato en la que se puede observar: el espaciado entre las líneas, la situación de los tabuladores y la longitud de la línea.

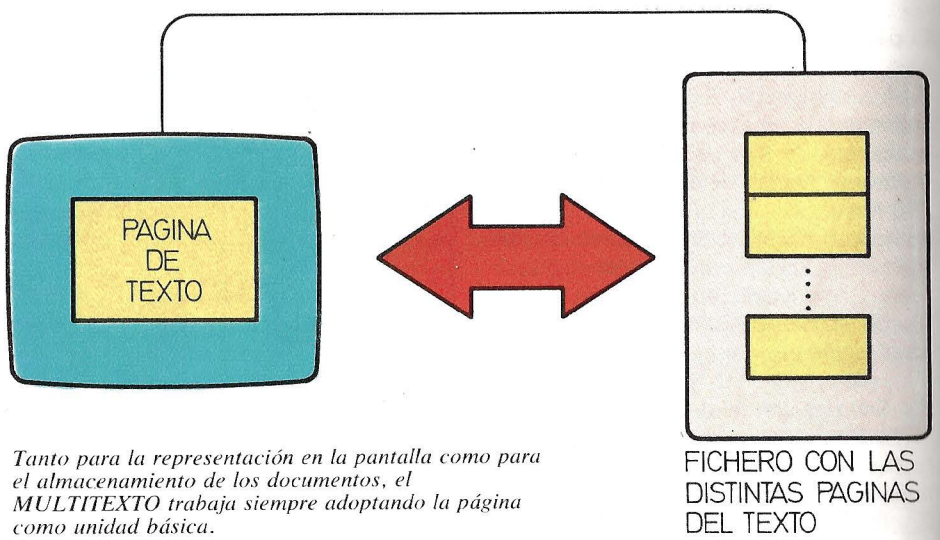
El límite de caracteres en cada línea es de 156, aunque en la pantalla tan sólo se podrán visualizar 80; en este caso, cuando el usuario haya introducido los primeros 80 caracteres, el programa desplazará automáticamente el documento hacia la izquierda, según se vayan añadiendo nuevos caracteres.

- *Símbolos en pantalla*

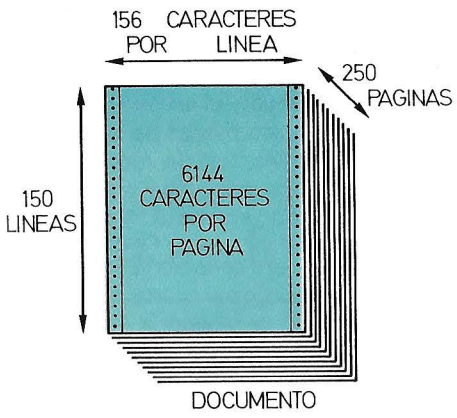
En algunos documentos se pueden introducir símbolos especiales que representan una situación no admisible en la pantalla del ordenador, pero sí en la impresora. Por ejemplo, escribir una determinada palabra en negrita. En estos casos, al utilizar la función correspondiente del MULTITEXTO, ésta quedará identificada mediante símbolos en pantalla.



Para incrementar la velocidad de ejecución, es recomendable copiar los disquetes originales de la aplicación en un disco rígido; esta es una tarea que no puede hacerse extensiva al disco del sistema que se encuentra protegido frente a cualquier intento de copia.



Tanto para la representación en la pantalla como para el almacenamiento de los documentos, el MULTITEXTO trabaja siempre adoptando la página como unidad básica.



Dimensiones máximas de un documento creado con el paquete de aplicación MULTITEXTO.

- *Desplazamiento horizontal*

Así se denomina al movimiento realizado con el cursor, ya sea de derecha a izquierda o bien de izquierda a derecha. Cuando el cursor llega en su desplazamiento al final de una línea, salta automáticamente al principio de la siguiente. De forma análoga, al llegar al principio de una línea salta también automáticamente al final de la línea anterior. El cursor no puede ser desplazado a un área de la pantalla en blanco.

- *Desplazamiento vertical*

Movimiento del cursor de arriba hacia abajo o viceversa. En el caso de que todas las líneas de la pantalla se encuentren ocupadas, el movimiento vertical del cursor implica un desplazamiento de una línea cada vez en el mismo sentido en el que se mueva el cursor. Cuando se llegue al principio o al final de una página y prosiga el desplazamiento, el cursor pasará a la página anterior o posterior, respectivamente.

- *Control del movimiento del cursor*

El paquete MULTITEXTO ofrece diversas posibilidades de movimiento automático del cursor, de forma que sin más que pulsar a una o dos teclas el cursor cambiará de la posición en la que se encuentre a la nueva posición deseada por el usuario.

- *Sobreiluminación*

Cuando se desea mover, copiar, insertar o

Potenciales usuarios de procesadores de textos

El tratamiento de textos automatizado consiste, sencillamente, en sustituir los tradicionales métodos de escritura por aplicaciones mecanizadas. En consecuencia, y de forma general, cabe afirmar que cualquier persona que tenga necesidad de escribir es un usuario potencial de los procesadores de textos. Un intento de clasificación nos lleva a distinguir entre los siguientes segmentos de usuarios:

1. *Secretariado*

En primer lugar hay que citar a los usuarios tradicionales: los encargados de realizar labores de secretariado. El procesador de textos no sólo actuará como sustitutivo de la máquina de escribir, sino que también permitirá la automatización de algunas labores de archivo.

2. *Departamentos comerciales y de gestión*

El volumen de documentos escritos por los departamentos comerciales y de gestión suele ser moderado, aunque constante. Ello los convierte en candidatos para disponer de un sistema para el tratamiento de textos continuamente ocupado.

3. *Departamentos técnicos*

Los departamentos técnicos de una empresa no suelen plantear una constante necesidad de producir documentos. Aunque bien es cierto que en determinados momentos puntuales deben producir documentos de gran volumen; por lo tanto, suele ser recomendable que compartan un procesador de textos con otro segmento de usuarios.

4. *Profesionales independientes*

Una típica frase en boca de cualquier profesional independiente es: "Hago de todo, de secretaria, de director general, e incluso de botones". No cabe duda que este colectivo puede obtener un gran provecho del procesador de textos. Dentro de este ámbito hay que resaltar a los profesionales de la escritura: periodistas, escritores, free-lancers... Si para cualquier profesional libre los procesadores resultan útiles, para estos últimos son fundamentales e imprescindibles.

5. *Usuarios domésticos*

Por último, cabe hacer mención al entorno doméstico. Existen procesadores de texto incluso para los microordenadores más simples e inevolucionados, aunque sus funciones son muy limitadas. En cualquier caso siempre cabe la posibilidad de utilizar el equipo doméstico para elaborar la correspondencia típica de un hogar.

MULTITEXTO es un procesador de textos que corresponde a la versión en castellano del paquete MULTIMATE. Está destinado a los ordenadores personales IBM-PC y compatibles.

eliminar parte del documento tratado, se puede proceder a sobreiluminar la zona del texto afectada. De esta forma, la porción implicada se diferenciará claramente del resto del documento, al brillar más que el texto circundante.

FUNCIONAMIENTO DEL MULTITEXTO

En el próximo capítulo se detallarán con precisión las principales funciones del paquete MULTITEXTO. No obstante, en los próximos párrafos se establece una clasificación temática de funciones, derivada de las opciones que figuran en el menú principal de la aplicación.

1. *Editar un documento antiguo*
Sirve para activar algún texto previamente almacenado en los soportes de almacenamiento. Para elegir el fichero a editar, el usuario debe llamarlo por su nombre de fichero, coincidente con el asignado en el instante en el que fue almacenado.

2. *Crear un documento nuevo*
Evidentemente, en este caso el programa no realizará ninguna lectura de información de la memoria de masa; sencillamente, permitirá al usuario teclear el texto correspondiente y solicitará un nombre para archivarlo en el soporte de almacenamiento.

3. *Imprimir un documento*
Esta opción se utiliza para ordenar la escritura de alguno de los documentos almacenados. La impresión es controlada mediante una cola de trabajos pendientes de imprimir. Antes de enviar un documento a la cola de impresión, el usuario puede especificar las características de la escritura (tamaño de página, tipo de impresora, fecha y hora en la que debe realizarse la impresión del documento...)

4. *Opciones para controlar la impresión*
Si se selecciona esta cuarta opción del menú principal, el programa permitirá al usuario controlar y gestionar la cola de trabajos pendientes de impresión, pudiendo alterar el orden de escritura, retener algún listado, etc. En definitiva, mediante esta opción se puede modificar cualquier característica de impresión.

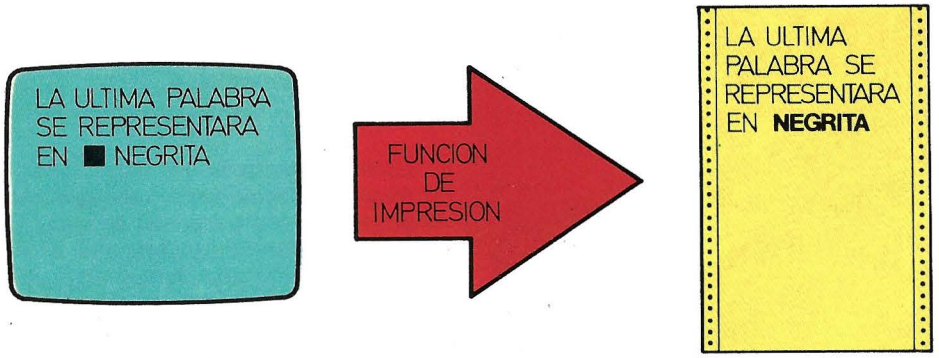
5. *Imprimir un documento fusión*
Permite mezclar en un único informe dos documentos introducidos de forma independiente. Esta facultad agiliza el trabajo con formularios, ya que se pueden diferenciar documentos especializados que al fusionarse producen el resultado deseado.

6. *Opciones para el manejo de documentos*
Como su propio nombre indica, este punto del menú principal sirve para mostrar al usuario un conjunto de opciones que le permitirán manejar los documentos de forma cómoda y eficiente.

7. *Prefijar parámetros generales del sistema*
En algunos casos, la aplicación MULTITEXTO se deja programar mediante determinados parámetros, de forma que cada usuario, en cada sesión de trabajo, pueda preseleccionar distintas características generales. La séptima opción del menú principal permite establecer los referidos parámetros.

8. *Revisar la ortografía de un documento*
Esta es una de las opciones más espectaculares del MULTITEXTO. Una vez seleccionada por el usuario y decidido el documento a revisar, el programa determinará los errores ortográficos cometidos en la introducción de los datos. Para acometer esta revisión, el programa se basará en la información almacenada en el disco de "diccionario".

9. *Volver al sistema operativo D.O.S.*
Opción para abandonar el trabajo con el MULTITEXTO. Al seleccionarla, el ordenador queda disponible para proceder a la ejecución de otras tareas.



Los símbolos de pantalla permiten especificar características que sólo quedarán patentes al imprimir el documento.

Otras estructuras de control

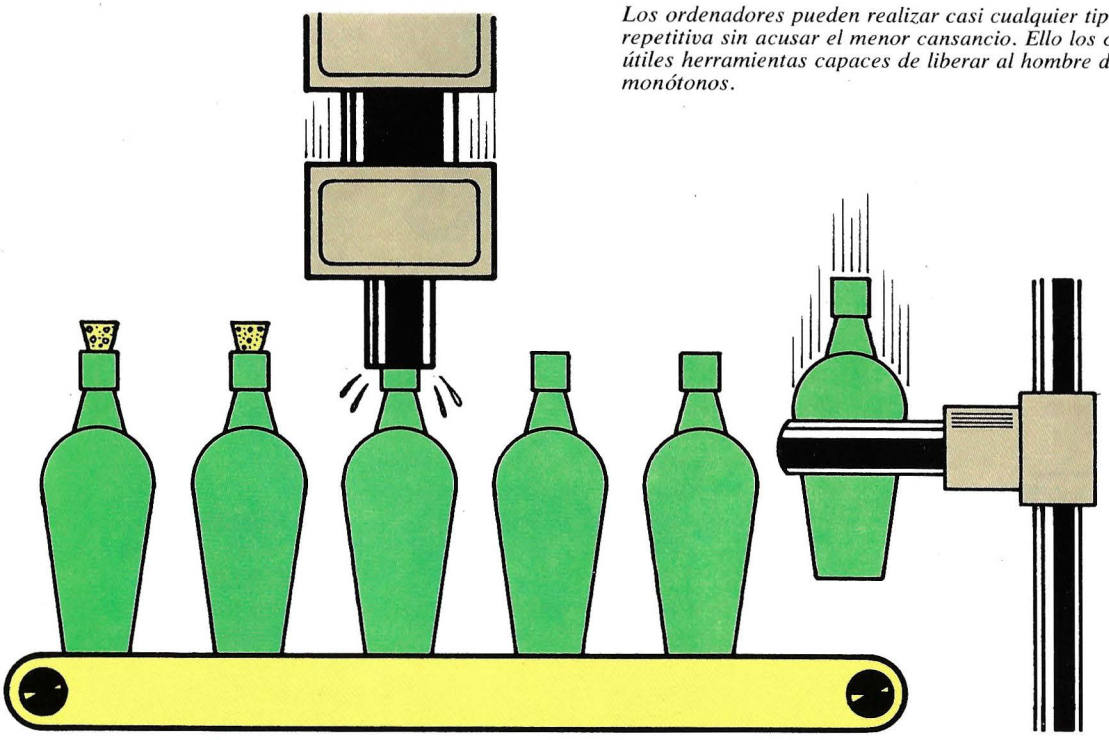
Nuevas formas de crear bucles

Los ordenadores son capaces de realizar un mismo trabajo un elevado número de veces. La ventaja de tal posibilidad de reiteración automática, es más que apreciable para el usuario, quien se ve descargado de la ardua y penosa repetición de trabajos reiterativos. Las máquinas no se cansan, ni abandonan su tarea... a menos que les falte el suministro de energía. ¿Y para qué están las máquinas, sino para liberar al hombre de esos trabajos monótonos y repetitivos?

En efecto, una de las capacidades más espectaculares de los ordenadores reside en la facultad de ejecutar iterativamente (repetitivamente) la serie de operaciones que se le indiquen. El control de tal actividad repetitiva está encomendado en el mundo de la informática, y más concretamente en el lenguaje BASIC, a las estructuras de control de tipo repetitivo; tal es el caso de FOR/NEXT. La función de estas estructuras de control es, precisamente, la de instruir a la máquina para que repita incansablemente una serie de instrucciones, hasta que se cumpla o se deje de cumplir una determinada condición; instante en el que se abandonará la tarea en curso y se continuará con otra diferente.

LA ESTRUCTURA WHILE/WEND

Cualquier versión del lenguaje BASIC, incluso los menos potentes, incorpora una serie de instrucciones de control: IF/THEN/ELSE, FOR/NEXT, GOTO, ... Todas ellas han sido objeto de estudio en capítulos anteriores. Sin embargo, algunos traductores BASIC van más lejos, y



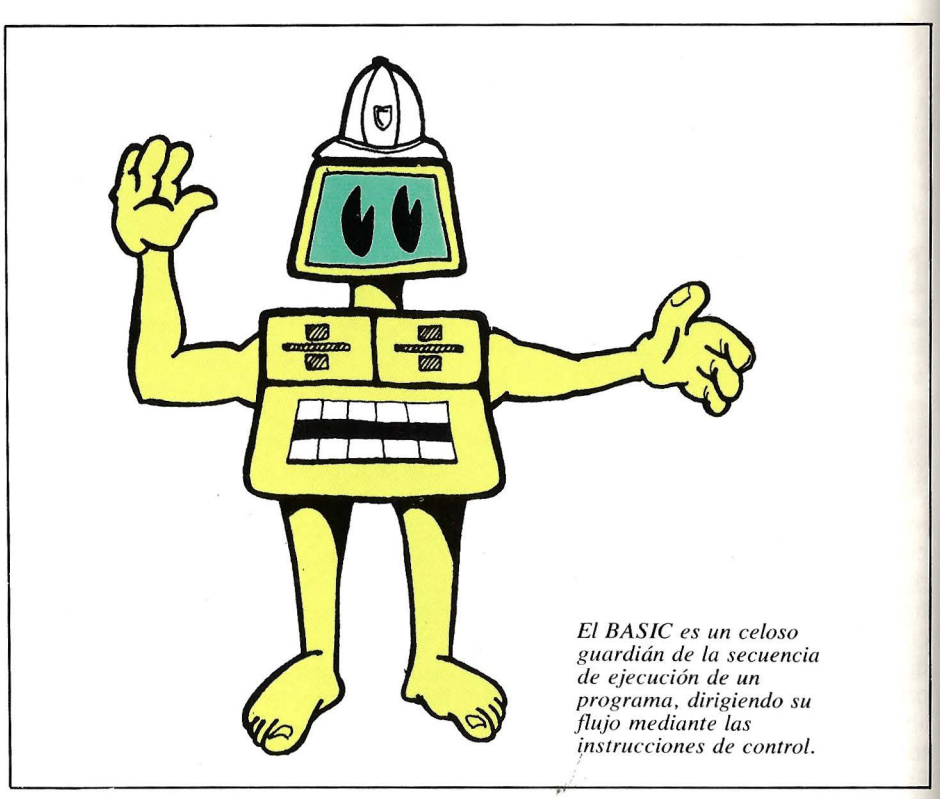
Los ordenadores pueden realizar casi cualquier tipo de tarea repetitiva sin acusar el menor cansancio. Ello los convierte en útiles herramientas capaces de liberar al hombre de los trabajos monótonos.

ofrecen ciertas instrucciones de control alternativas más versátiles y manejables, cuyo estudio se acometerá en el presente capítulo.

La incorporación de estas nuevas sentencias se debe al intento de aproximar un poco más el BASIC a los lenguajes de programación estructurados. Estos basan su eficacia en el uso exclusivo de tres estructuras básicas: secuencial, selectiva y repetitiva, apoyadas en potentes instrucciones propias de los lenguajes estructurados y para las que el BASIC no tiene réplica en la mayor parte de los casos.

Una de ellas, que aún siendo poco frecuente forma parte del repertorio de instrucciones BASIC de muchos ordenadores personales, es la instrucción WHILE/WEND, cuyo formato más general es el siguiente:

<N1.> WHILE <expr.>
<bucle>
<N1.> WEND



El BASIC es un celoso guardián de la secuencia de ejecución de un programa, dirigiendo su flujo mediante las instrucciones de control.

La instrucción WHILE hará que se repita una determinada acción mientras sea cierta la condición impuesta.

Esta instrucción permite la ejecución repetitiva de un conjunto de sentencias englobadas en un bucle, delimitado por las palabras clave WHILE (por delante) y WEND (al final).

Como se recordará, en las instrucciones FOR/NEXT se repetía el bucle un número de veces determinado de antemano por los valores inicial y final de su variable de control. En esta nueva instrucción, el número de veces que se ejecutará el bucle está indeterminado en un principio. El bucle será ejecutado reiteradamente mientras (WHILE) sea cierta la expresión <expr.> que lo controla; esta última será una expresión de tipo booleano. En el instante en el que dicha expresión deje de adoptar el valor lógico 1 (verdadero), el bucle será abandonado, pasando la secuencia de ejecución del programa a la instrucción situada inmediatamente detrás de la palabra WEND.

El ejemplo que sigue contribuirá a clarificar la diferencia existente entre las instrucciones WHILE/WEND y FOR/NEXT:

WHILE/WEND

Repetición de un bucle de sentencias, mientras sea cierta la condición que lo controla.

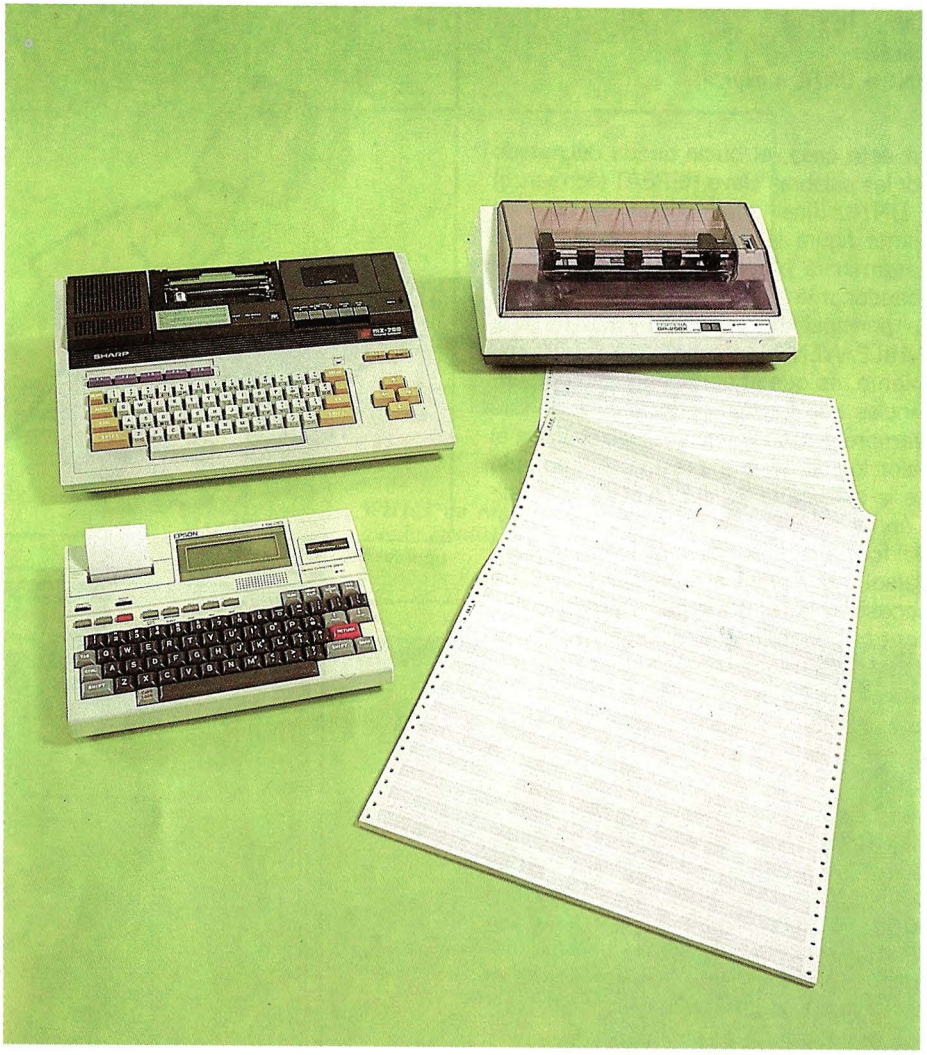
Formato: *<N1.> WHILE <expr.> <bucle> WEND*

Ejemplo:
```
10 WHILE I<0
20 I=I+1
30 WEND
```

```
100 CONTROL=1
110 WHILE CONTROL
120 CONTROL=0
130 FOR I=1 TO 19
140 IF A$(I)>A$(I+1)
    THEN SWAP A$(I), A$(I+1): CONTROL=1
150 NEXT I
160 WEND
■
```

El objetivo es clasificar en orden creciente los valores que se suponen ya almacenados en una matriz de caracteres A$, previamente dimensionada con 20 elementos. El bucle FOR del ejemplo se ejecutará 19 veces, comprobando si un elemento es menor que el siguiente, en cuyo caso se procederá a intercambiar sus posiciones respectivas mediante el comando SWAP. En el bucle WHILE, en cambio, no está determinado en principio el número de veces que se ejecutará; la reiteración proseguirá hasta que estén ordenados todos los elementos de la matriz A$.

La variable que señalará el final de la repetición del bucle es la denominada CONTROL. Su valor lógico se pone a uno (verdadero) cada vez que se efectúa un cambio de orden, para así volver a comprobar si la nueva sucesión de valores de A$ es ya correcta. Como se deduce fácilmente, es necesario modificar el valor de la expresión de control asociada a WHILE dentro de su propio bucle. De no ser así, el bucle no se ejecutaría nunca si la condición impuesta resultara falsa al llegar al mismo. Aunque también podría suceder lo contrario: la ejecución podría eternizarse en un bucle infinito, si la condición resultara cierta al entrar en el bucle WHILE. En el siguiente ejemplo jamás podrá abandonarse la ejecución repetida del bucle WHILE/WEND, puesto que la condición no es actualizada dentro del propio bucle:



La capacidad del ordenador para ejecutar tareas repetitivas es una de las facultades que más han potenciado su implantación actual.

```
200 INDI=1
210 WHILE INDI
220 PRINT "PULSA BREAK, POR FAVOR!"
230 WEND
■
```

De ahí que en el primer ejemplo se asignara el valor 0 a la variable CONTROL en la línea 120: dentro del bucle. Con ello será posible salir del mismo. El valor de CONTROL sólo tomará el contenido lógico 1 —en orden a repetir el bucle—, cuando se efectúe un cambio en el orden de dos o más elementos en la línea 140.

Los bucles WHILE/WEND pueden ser anidados unos dentro de otros, de forma similar a como se anidan los bucles FOR/NEXT o las sentencias IF/THEN/ELSE. En este caso, cada palabra WEND cerrará el bucle correspondiente a la instrucción WHILE que se haya ejecutado con mayor inmediatez. Así, el último WEND corresponderá a la sentencia WHILE en cuyo bucle estén inmersas todas las restantes sentencias WHILE anidadas. Otra característica importante de la sentencia WHILE/WEND es que el bucle no se ejecutará ninguna vez si al llegar a ella la expresión que lo controla tiene el valor lógico 0 (falso). El motivo hay que buscarlo en el hecho de que la condición se evalúa antes de entrar en el bucle, y no tras entrar en el mismo, en cuyo caso siempre se ejecutaría el bucle al menos una vez.

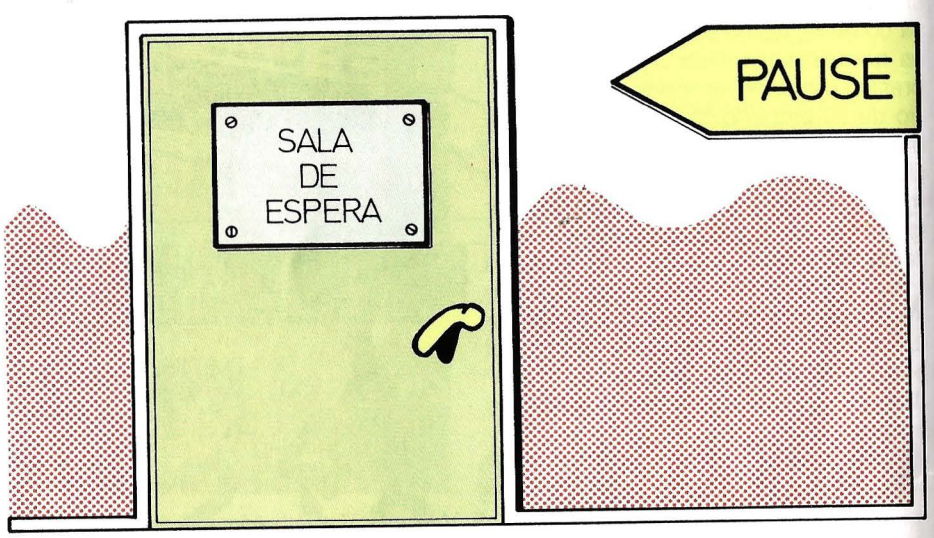
REPETICIONES INAGOTABLES

Otra estructura de control, también presente en varios intérpretes BASIC, aunque menos habitual que la WHILE/WEND, es REPEAT/UNTIL. Raro es el traductor BASIC que posee ambas estructuras, de tal forma que si tiene una de ellas, la otra, generalmente, estará ausente. Su formato, muy similar al de la estructura WHILE/WEND, es el siguiente:

<N1.> REPEAT
<bucle>
<N1.> UNTIL <expr.>

En este caso, el bucle queda delimitado por las palabras clave REPEAT (comienzo) y UNTIL (final). A continuación de esta última figura la expresión <expr.>, que determinará el instante en que se debe abandonar la ejecución del bucle.
La principal diferencia con la estructura WHILE/WEND, en cuanto a su funcionamiento, es que ahora el conjunto de sentencias que forman el bucle se ejecutará siempre al menos una vez, sea cual sea el valor lógico de la expresión. Al contrario de la estructura WHILE/WEND, la condición se evalúa al llegar al final del bucle; de forma que si tiene el valor lógico 0 (falso), se volverá a repetir el bucle. Así sucesivamente hasta que la expresión de control adopte el valor lógico 1 (verdadero), en cuyo caso el programa seguirá ejecutándose a partir de la instrucción cuyo número de línea sigue al que corresponde a la línea que contiene la palabra UNTIL.
Un ejemplo de aplicación de esta estructura cabe situarlo en un programa de juegos, en el que la ejecución del mismo no comienza hasta que se haya pulsado una tecla cualquiera. Para ello, se explorará el teclado repetidamente, mediante el comando INKEY$, hasta que se detecte la pulsación de alguna tecla. Esta simple instrucción conformará el bucle de la estructura iterativa.
La condición a evaluar cada vez que se ejecute el bucle es: ¿se ha pulsado ya alguna tecla? En caso afirmativo, se abandonará el bucle y se ejecutará el resto del programa. Al efecto se puede utilizar una variable inicializada con el valor lógico 0 (falso), pasando a asignarle el valor 1 (verdadero) en el momento en el que se cumpla la condición impuesta.

En el ejemplo que figura en el texto, el ordenador aguardará a que sea accionada una tecla para empezar la ejecución de la zona principal del programa.



La instrucción PAUSE introduce al ordenador en una "sala de espera", que no abandonará hasta que transcurra el tiempo indicado.

REPEAT/UNTIL

Repetición de un bucle de sentencias, hasta que sea cierta la condición que lo controla.

Formato: *<N1.> REPEAT <bucle> UNTIL <expr.>*

Ejemplo:

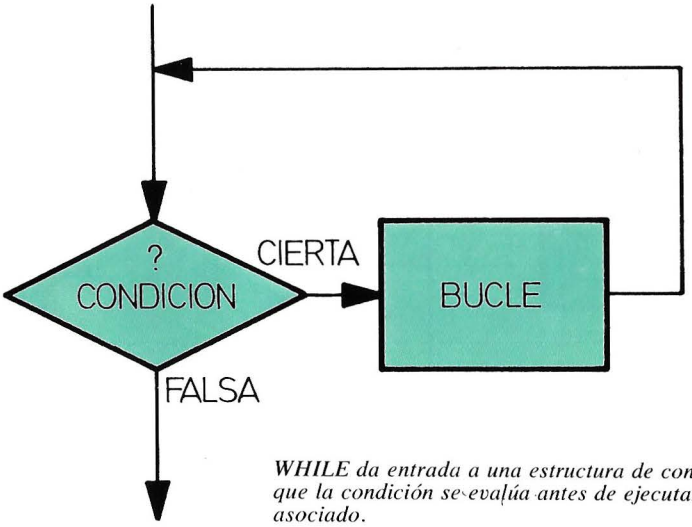
```
10 REPEAT
20 READ D
30 PRINT D;
40 UNTIL D="*"
50 DATA H,O,L,A,*
```

Esta se verificará al pulsar una tecla, lo que se detectará cuando el valor obtenido por INKEY$ sea distinto de " " (cadena vacía) que es el valor normal de vuelta cuando no se ha accionado tecla alguna.
El listado del programa capaz de realizar tal cometido puede coincidir con el siguiente:

```
10 PRINT "PULSE UNA TECLA CUALQUIERA
   PARA EMPEZAR"
20 YAESTA=0
30 REPEAT
40 IF INKEY$< >" " THEN YAESTA=1
50 UNTIL YAESTA
60 REM AQUI EMPIEZA EL PROGRAMA
   PRINCIPAL
```

La variable YAESTA será la que indique el momento en el que debe ser abandonado el bucle: cuando su valor lógico sea 1. De inmediato, se pasará a ejecutar la línea 60 en la que dará comienzo el programa de juego.
Dado que esta estructura permite que se ejecute el bucle al menos una vez, no hay peligro de que el ordenador quede encerrado en un bucle sin salida al asignar a YAESTA el valor lógico 0 en la línea 20 del bucle.

MAS Y MAS LAZOS...



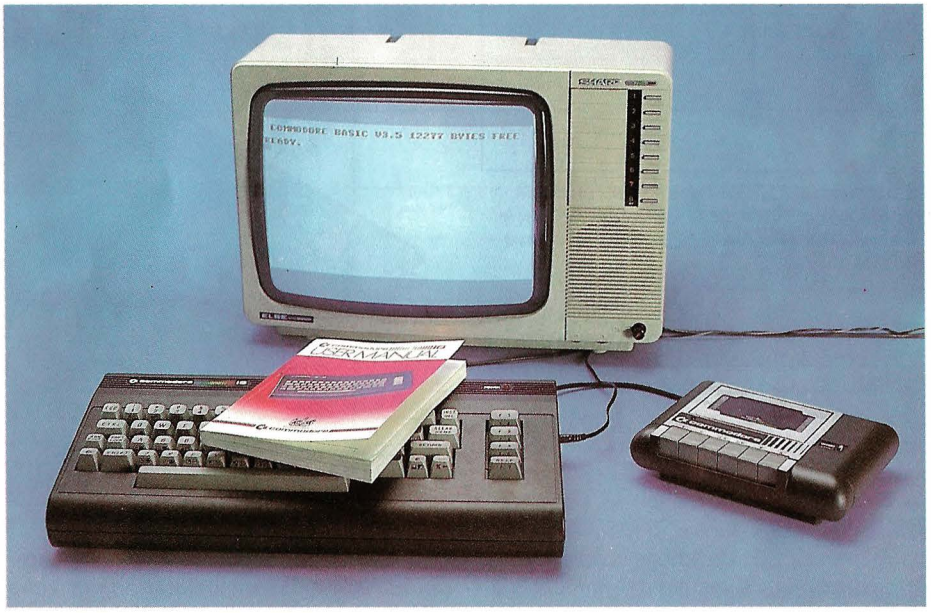
WHILE da entrada a una estructura de control en la que la condición se evalúa antes de ejecutar el bucle asociado.

Una sentencia BASIC muy similar a WHILE/WEND y REPEAT/UNTIL, aunque menos frecuente aún en el variado repertorio de intérpretes BASIC, es la instrucción DO/LOOP. Esta puede considerarse como un compendio de la WHILE/WEND y de la REPEAT/UNTIL, por el motivo que se verá más adelante.

Su formato es el siguiente:

<N1.> DO [WHILE <expr.>]
<bucle>
<N1.> LOOP [UNTIL <expr.>]

Ahora, la rutina que constituye el bucle está delimitada por las palabras clave DO como principio y LOOP como final. La salida del bucle se controla, al igual que en los casos anteriores, por medio del valor lógico que toma una determinada expresión. Hasta aquí no hay novedad alguna.
La diferencia radica en que mientras en la estructura WHILE/WEND la condición para abandonar el bucle se evalúa al principio, y en la REPEAT/UNTIL al final del mismo, en la estructura DO/LOOP la condición puede ser chequeada en cualquiera de los dos puntos, o incluso en ambos. Por esta razón se afirmaba anteriormente que DO/LOOP puede considerarse como una extensión de las otras dos estructuras, lo que le confiere una mayor potencia.
La expresión que con su valor lógico determina cuándo debe ser abandonado el bucle, puede ir situada como argumento de una función WHILE (mientras que...), o



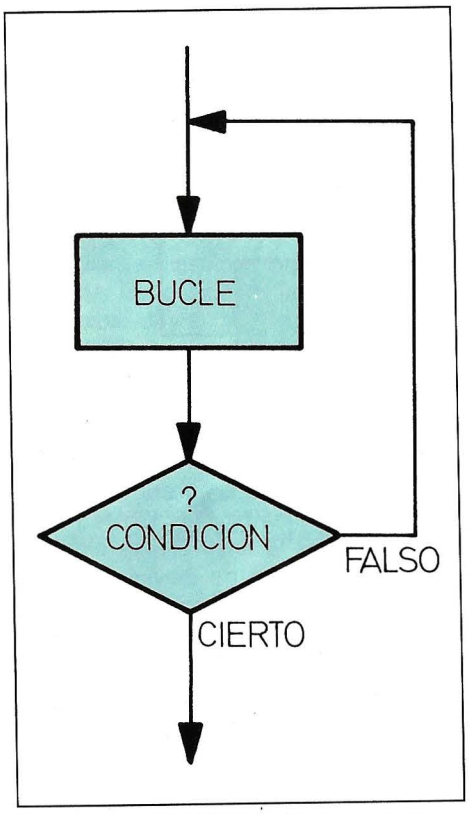
El reciente COMMODORE 16 es uno de los pocos equipos domésticos cuyo intérprete BASIC incluye la instrucción de control DO/LOOP.

DO/LOOP

Repetición de un bucle de instrucciones, mientras o hasta que sean ciertas las condiciones que lo controlan.

Formato: <N1.> DO (WHILE <expr.1>) <bucle> LOOP (UNTIL <expr.2>)

Ejemplo:
```
10 DO WHILE FLIP<>FLOP
20 IF FLIP>FLOP THEN FLIP=FLIP-FLOP
     ELSE FLOP=FLOP-FLIP
30 LOOP UNTIL FLOP>FLIP
```



*La estructura **REPEAT** evalúa la condición establecida al finalizar la ejecución del bucle. Ello significa que éste se ejecutará al menos una vez.*

bien como argumento de una función UNTIL (hasta que...). Estas pueden emplazarse detrás de la palabra DO que abre el bucle, o detrás de la palabra LOOP que lo cierra, o incluso en ambas. De esta forma se conseguirá salir del bucle por cualquiera de los dos puntos, dependiendo de los valores lógicos de ambas condiciones. Esta característica resulta de gran utilidad para evaluar a la vez dos condiciones distintas, de forma que al cumplirse una cualquiera de ellas se abandone la repetición del bucle.

```
 5 INPUT NUM
10 MIN=10:MAX=50
20 DATA 15,13,-20,5-11,18,25,-15,4,1000
30 DO WHILE NUM>MIN
40 READ D
50 PRINT NUM
60 NUM=NUM+D
70 LOOP UNTIL NUM>MAX
80 PRINT "FIN"
```

La ejecución del listado anterior, al introducir por el teclado el valor 11 como asignación de NUM, mostrará en la pantalla los siguientes valores aleatorios comprendidos entre 10 y 50:

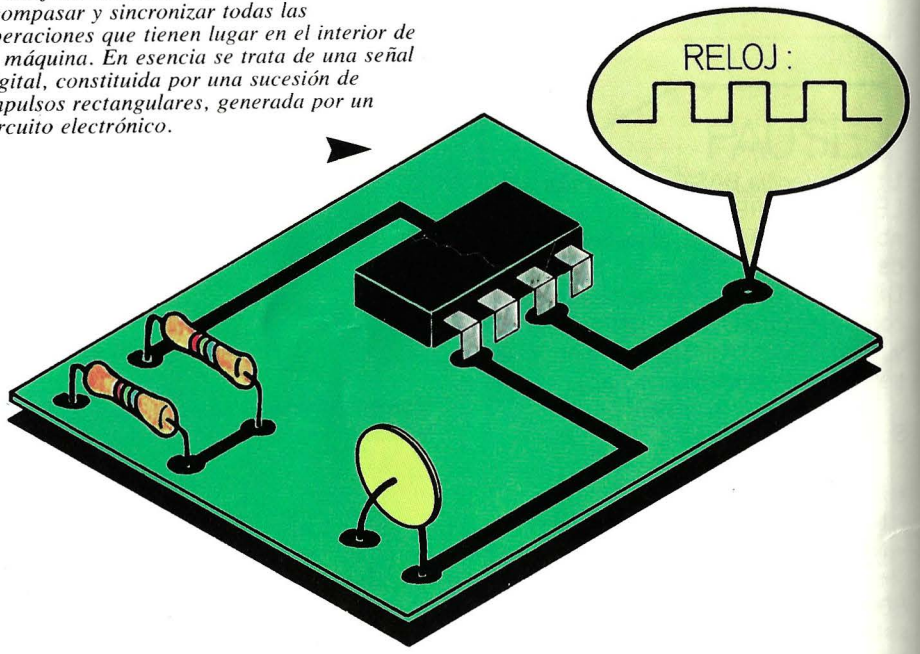
```
11
26
39
19
24
13
31
FIN
■
```

El bucle se ha visto interrumpido al cumplirse la condición incluida en la zona LOOP: la variable NUM ha alcanzado un valor superior al máximo permitido (MAX). Introduciendo otros valores para NUM, puede resultar que el lazo se interrumpa al verificarse la condición de la zona DO.

LA SALA DE ESPERA DEL BASIC

El ordenador fundamenta su gran eficacia en su naturaleza de máquina programable, idónea para multitud de tareas diferentes, y además en la gran velocidad con la que es capaz de realizar la tarea encomendada. No obstante, en algunas ocasiones puede ocurrir que no sea necesario explotar su gran velocidad, o incluso que se desee ralentizar las acciones del ordenador para conseguir determinados efectos; por ejemplo, que dé tiempo al usuario para realizar una determinada acción (ver detenidamente los resultados de un problema visualizados en la pantalla...).
Para ralentizar su actividad, existe una primera alternativa, obvia e inmediata: utilizar la sentencia FOR/NEXT de forma que "se repita" un número determinado de veces la "acción" de "no hacer nada":

El reloj del ordenador se encarga de acompasar y sincronizar todas las operaciones que tienen lugar en el interior de la máquina. En esencia se trata de una señal digital, constituida por una sucesión de impulsos rectangulares, generada por un circuito electrónico.




```
100 INPUT "INTRODUCE DOS NUMEROS";A,B
110 PRINT "HUM...DEJAME VER"
120 FOR I=1 TO 500:NEXT I
130 PRINT "CREO QUE LA SUMA DE ";A;" Y
    ";B;" ES: ";
140 FOR I=1 TO 1000:NEXT I
150 PRINT A+B
```

Según el valor final que se le asigne a la variable de control del bucle FOR, el tiempo "muerto" durante el que el ordenador parece estar pensando la solución será más o menos dilatado. En la línea 120 del ejemplo anterior, el retardo introducido es menor (la mitad) que el retardo introducido en la línea 140.

TABLA DE CONVERSION

ORDENADOR	WHILE/WEND	REPEAT/UNTIL	DO/LOOP	PAUSE
	WHILE <e1>: WEND	REPEAT: UNTIL <e1>	DO {WHILE <e1>} LOOP {UNTIL <e2>}	PAUSE <exp.>
APPLE II (APPLESOFT)	—	—	—	—
APRICOT (M-BASIC)	—	—	—	PAUSE <exp.>
ATARI	—	—	—	—
CBM 64	—	—	—	—
DRAGON	—	—	—	—
EQUIPOS MSX	—	—	—	PAUSE <exp.>
HP-150	WHILE <e1>: WEND	—	—	—
IBM PC	WHILE <e1>: WEND	—	—	PAUSE <exp.>
MPF	—	—	—	—
NCR DM-V (MS-BASIC)	WHILE <e1>: WEND	—	—	—
NEW BRAIN	—	—	—	—
ORIC	—	REPEAT: UNTIL <e1>	—	—
SHARP MZ-700 (MZ-BASIC)	—	—	—	—
SINCLAIR QL	—	REPEAT: ENDREPEAT <e1>	—	PAUSE <exp.>
SPECTRAVIDEO	—	—	—	—
ZX-SPECTRUM	—	—	—	PAUSE <exp.>

Este tipo de retardos es tan útil en multitud de ocasiones, que la mayoría de los intérpretes BASIC poseen un comando específico cuya única misión es exclusivamente ésa. La forma más general de este comando es:

PAUSE <expr.>

La actividad de todo ordenador es sincronizada y acompasada por las evoluciones de un reloj interno. Este se reduce a una señal digital, de forma cuadrada, creada por unos circuitos digitales específicos; circuitos que constituyen el pequeño corazón del ordenador, que le da vida con sus pulsaciones. La expresión <expr.> que sigue a la palabra PAUSE (pausa), está relacionada con dicho reloj interno: dependiendo del valor que se obtenga al evaluar la expresión, el tiempo que la máquina permanecerá "a la espera" será mayor o menor.

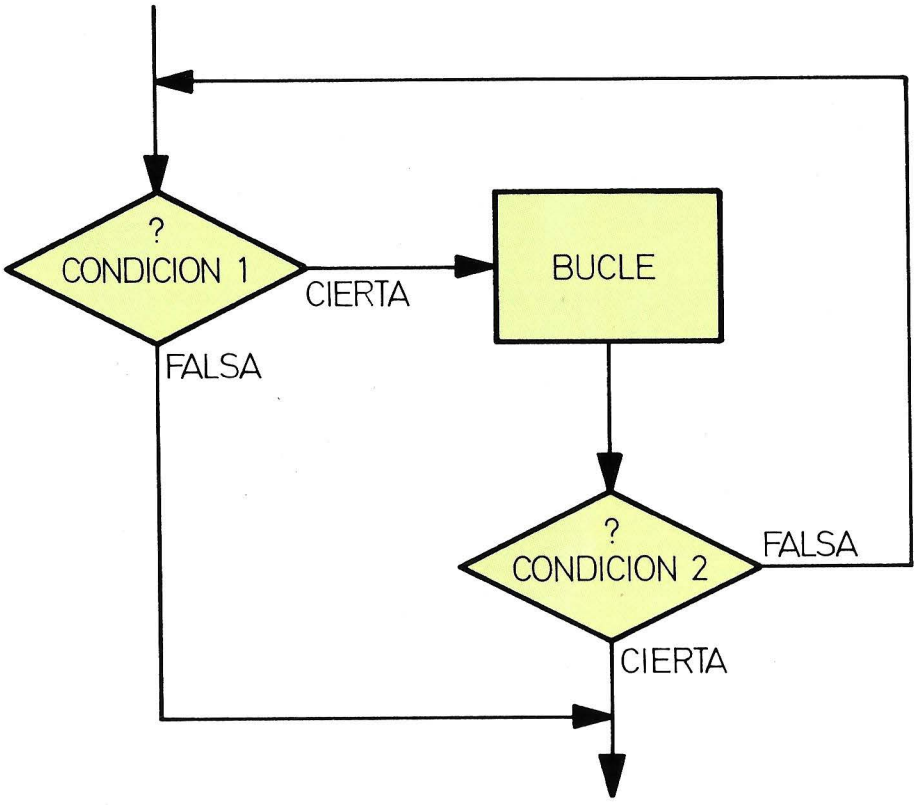
Al igual que el pulso de una persona puede ser distinto al de otra, dependiendo de su constitución física, así también ciertos ordenadores tendrán un pulso más o menos rápido que otros, condicionando la propia velocidad de trabajo del ordenador. Ello tiene una consecuencia directa en la instrucción PAUSE: para lograr una pausa de un segundo, el valor que hay que situar

PAUSE

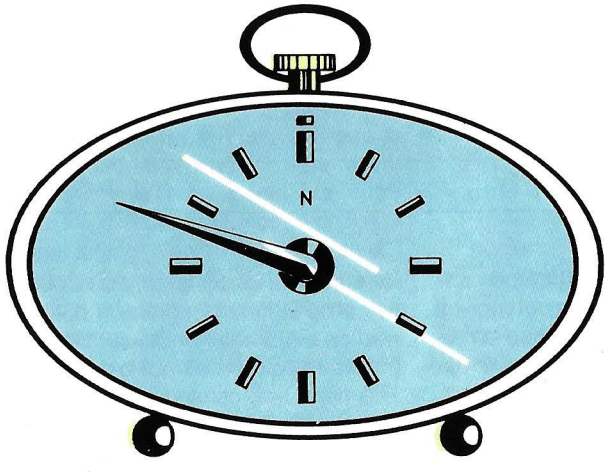
Detención temporal de la ejecución de un programa durante un tiempo especificado.

Formato: *<N1.> PAUSE <expr.>*

Ejemplos:
```
10 PAUSE 60
100 PAUSE N+10
```



La estructura de control DO/LOOP sintetiza las características propias de las estructuras WHILE y REPEAT.

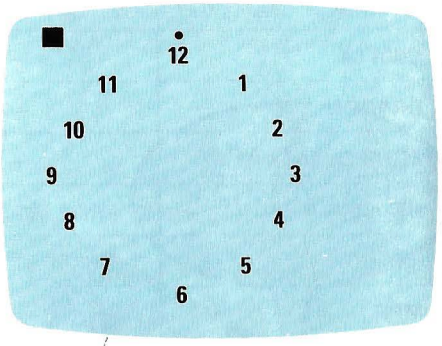


Mediante el programa incluido en el texto, la pantalla del ordenador se convierte en un cronómetro que va marcando el paso de los segundos.

detrás de la palabra PAUSE será muy diferente de unos ordenadores a otros. Así, por ejemplo, para obtener una pausa de 1 segundo, en unos casos habrá que hacer PAUSA 1, mientras que en otros será necesario poner PAUSA 60.
Una vez determinado el valor adecuado para obtener una pausa de un segundo, el cálculo del valor necesario para cualquier otro retardo se reduce a una simple multiplicación.
El siguiente ejemplo ilustra el empleo de la instrucción PAUSE:

```
 5 PI=3.14159
10 FOR H=1 TO 12
20 PRINT AT 20+9*SIN(H/6*PI),
   11-9*COS(H/6*PI);H
30 NEXT H
40 FOR S=0 TO 10000
50 LET G=S/30*PI
60 LET X=20+10*SIN(G)
70 LET Y=11-10*COS(G)
80 PRINT AT X, Y;"."
90 PAUSE 60
100 PRINT AT X, Y;" "
110 NEXT S
```

Al ejecutar el programa, aparecerá en la pantalla el siguiente resultado:



En efecto, el programa tiene como objeto dibujar en la pantalla un reloj que mida los segundos. Su correcto funcionamiento en distintos ordenadores se verá limitado por el hecho, comentado anteriormente, de los distintos valores aplicables a la sentencia PAUSE.
El valor de PAUSE de la línea 90 debe ser modificado en cada caso concreto para hacer que el reloj evolucione correctamente. Asimismo, será necesario en ciertos casos modificar la escala de las posiciones en pantalla para la sentencia PRINT AT, así como la inclusión de la constante PI (3,1416).
Hay que tener en cuenta el tiempo invertido en realizar el cálculo de los valores de G, y el tiempo empleado en imprimir en la pantalla el signo ".", y borrarlo. Ello significa que el tiempo muerto consumido por la sentencia PAUSE no debe ser exactamente de un segundo si se quiere obtener un reloj de apreciable exactitud. Será preciso cronometrarlo con un reloj convencional, e ir ajustando el valor de PAUSE hasta su valor correcto.
Otra característica de la sentencia PAUSE es que puede inhibirse al pulsar una tecla cualquiera del teclado. Con ello se puede lograr el efecto de arrancar un programa al cabo de un tiempo determinado, o bien en el instante en el que alguien accione una tecla cualquiera.

Pascal (6)

Los procedimientos del Pascal

A la hora de confeccionar un programa, se presenta con frecuencia la necesidad de utilizar repetidamente, y en distintos puntos del mismo, un mismo bloque de instrucciones; instrucciones que realizan una tarea específica, coincidente en los distintos casos. Ello obliga a escribir el mismo conjunto de sentencias, tantas veces como sea necesaria su presencia a lo largo del programa. Este bloque de sentencias suele recibir el nombre de subprograma. En PASCAL, cualquier subprograma es posible escribirlo una sola vez, y activarlo mediante una simple llamada al mismo cuando sea necesaria su ejecución.

Un programa es un edificio de compleja realización. Por ello, es importante que esté bien estructurado y sea fácil de leer; y no sólo por el traductor de lenguaje, sino también por el usuario o el programador. En base a ello, los modernos lenguajes de programación —tal es el caso del PASCAL—, tienden a ser estructurados y modulares. Sin lugar a dudas, el mantenimiento o modificación de un programa, será más sencillo si éste está dividido en pequeñas porciones (módulos) independientes.

En PASCAL, un subprograma se denomina PROCEDURE (procedimiento). El conjunto de sentencias que lo constituyen se crean por medio de la "declaración del procedimiento". En ella se asocia un identificador al procedimiento, por medio del cual será activado en el instante apropiado.

La declaración de un procedimiento adopta la misma estructura que un programa PASCAL; razón por la que se puede considerar a un programa como un verdadero "superprocedimiento" PASCAL. En consecuencia, el procedimiento constará de una serie de parámetros de entrada/salida opcionales, de una zona de declaración de variables, también opcional, y finalmente de un cuerpo de sentencias ineludible, englobado dentro de un bloque BEGIN/END. La declaración de un procedimiento debe ser posterior a la propia declaración de variables, por medio de VAR, del programa principal.

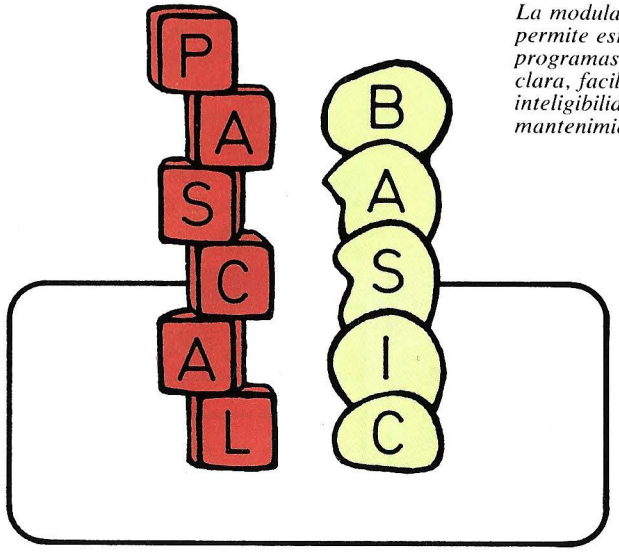
PROCEDIMIENTOS SIN PARAMETROS

Como ya se mencionó, los parámetros de entrada/salida de un procedimiento son opcionales; es posible definir procedimientos sin parámetros y también procedimientos con parámetros.

Los procedimientos sin parámetros tienen como finalidad realizar una determinada acción en la que no se producirá ningún resultado aprovechable por otra zona del programa; a su vez, el procedimiento sin parámetros tampoco necesitará ningún dato elaborado por el resto del programa. El siguiente ejemplo ilustra la declaración de un procedimiento sin parámetros. Más adelante se analizarán con más detalle cada una de sus zonas constitutivas, así como su cometido.

```
PROCEDURE subrayar;
        VAR n:INTEGER;
        BEGIN
        FOR n:=1 TO final DO
                WRITE ('-');
        WRITELN
        END;
```

La estructura de la declaración es, tal y



La modularidad del PASCAL permite estructurar los programas de forma limpia y clara, facilitando su inteligibilidad y, por ende, su mantenimiento.

como se adelantó, totalmente similar a la de un programa. Tan sólo varía su cabecera: en lugar de utilizar la palabra clave PROGRAM, se emplea ahora la palabra PROCEDURE.

La palabra que identifica a este procedimiento es "subrayar", la cual debe ser un identificador válido en PASCAL. Su cometido será activar el procedimiento en el punto adecuado del programa, además de dar una idea del objetivo del procedimiento; esto último siempre que el nombre utilizado resulte sugerente. El procedimiento del ejemplo tiene como misión visualizar en el dispositivo de salida (la pantalla), una línea de guiones, que actuará como subrayado de la línea que se haya impreso con anterioridad.

La zona de declaraciones de un procedimiento es totalmente análoga a la de un programa. De nuevo, su misión es la de declarar las variables que vayan a ser utilizadas únicamente dentro del procedimiento. En el caso de la variable "final", su valor le será asignado fuera del procedimiento y, por lo tanto, no es necesaria su declaración dentro del mismo, aunque sí está permitido el referirse a ella. La variable "n" se dice que es una variable local, que sólo podrá ser referida dentro de ese procedimiento, mientras que la variable "final" es una variable global, que puede ser utilizada en cualquier zona del programa. Por lo tanto, hay que tener en cuenta que un mismo identificador, declarado como variable global a nivel de programa y a la vez como variable local en un procedimiento, no representará a la misma variable, recibiendo valores diferentes en cada zona del programa.

Finalmente, el cuerpo de sentencias encerrado entre las palabras BEGIN y END, incluirá al conjunto de sentencias ejecutables que se encargarán de realizar la misión encomendada al procedimiento. En principio, puede parecer un contrasentido el hecho de incluir sentencias ejecutables en la zona de declaraciones del programa; ello se puede interpretar como una indicación de la acción a realizar al ser activado el procedimiento.

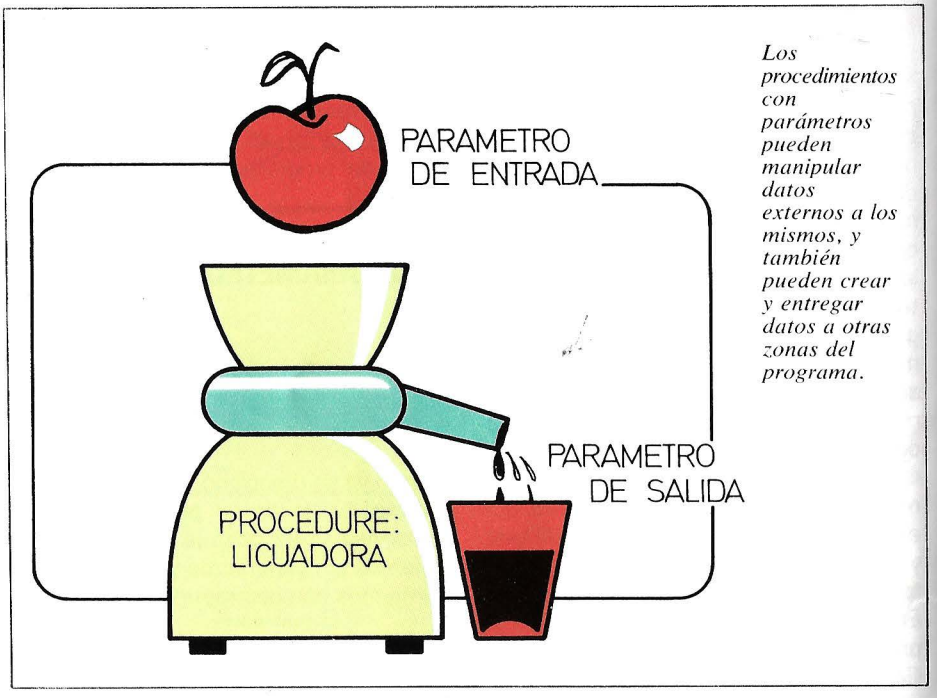
Para activar un procedimiento sin parámetros basta tan sólo con escribir la palabra o identificador que se colocó detrás de la palabra PROCEDURE. El siguiente programa utiliza el procedimiento creado en el ejemplo anterior:

```
PROGRAM cabecera (INPUT,OUTPUT);
VAR pag,final:INTEGER;
PROCEDURE subrayar;
      VAR n:INTEGER;
      BEGIN
            FOR n:=1 TO final Do
            WRITE('-');
            WRITELN
      END;
BEGIN
      WRITELN('HASTA QUE COLUMNA');
      READLN(final);
      WRITELN('SUBRAYADO HASTA LA
      COLUMNA:',final)
      subrayar
END.
```

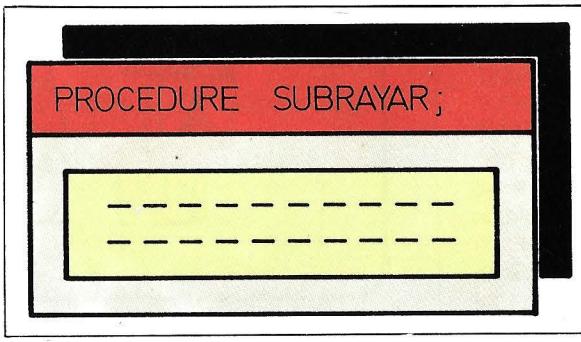
La sentencia de llamada que activa al procedimiento es su identificador "subrayar", tal y como se observa en el ejemplo. Ello desencadena la ejecución de las acciones indicadas en el bloque de sentencias. Si la ejecución desea repetirse en cualquier otra zona del programa, bastará con hacer una nueva llamada al procedimiento en cuestión utilizando su identificador: "subrayar".

PROCEDIMIENTOS CON PARAMETROS

Cuando se desee que un determinado subprograma pueda manejar datos creados en otra zona del programa, o incluso crear sus propios datos para utilizarlos en el resto del programa, habrá que recurrir a los denominados procedimientos con parámetros. El término "parámetros" se refiere a esos datos que manipula el procedimiento y que tienen una procedencia o



Los procedimientos con parámetros pueden manipular datos externos a los mismos, y también pueden crear y entregar datos a otras zonas del programa.



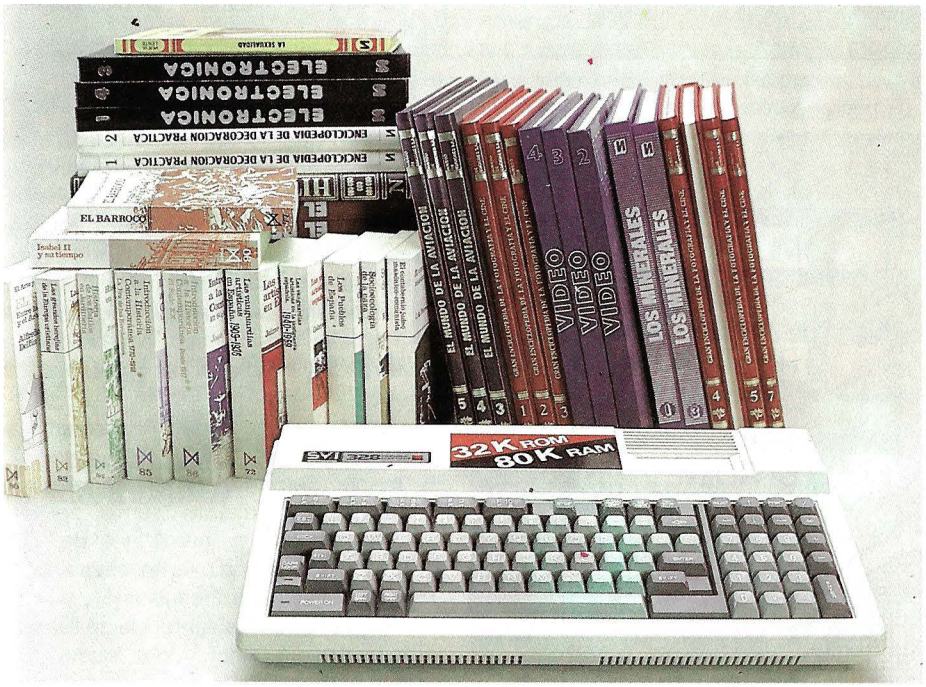
Un procedimiento sin parámetros no produce resultados de salida ni exige el suministro de datos de entrada.

destino exterior al procedimiento. En consecuencia, los parámetros pueden ser de entrada, de salida o de entrada y salida a la vez. Estos parámetros se indicarán encerrados entre paréntesis, detrás del identificador del procedimiento; algo semejante a lo que se hace en la cabecera del programa. Esta es la única diferencia sustancial respecto a los procedimientos sin parámetros. La forma más general de un procedimiento con parámetros es la siguiente:

PROCEDURE <identificador> (<param.>
[;<param.>;...]);
<zona de declaraciones>;
<bloque de sentencias>.

Los parámetros reflejados en la cabecera del procedimiento se llaman *parámetros formales,* debido a que en el momento de declarar el procedimiento no tienen asignado ningún valor concreto. El valor se asignará al hacer la llamada al procedimiento, tal y como analizaremos más adelante. Los parámetros formales de entrada del procedimiento adoptarán la forma siguiente:

identificador <:> tipo>



Los modernos lenguajes de programación —tal es el caso del PASCAL— tienden a ser estructurados y modulares. Ello facilita la confección y el mantenimiento de los programas para el tratamiento de información.

Los parámetros actuales de la sentencia de llamada transfieren los datos a los parámetros formales del procedimiento.

En este caso, al ser tan sólo un parámetro de entrada, su valor no estará disponible fuera del procedimiento; esto es: no podrá coincidir con un resultado del mismo. En cambio, los parámetros formales de entrada y salida, adoptarán un nuevo aspecto:

VAR <identificador>:<tipo>

Ahora, se obtendrá en este parámetro un resultado del procedimiento que puede ser utilizado en cualquier otra zona del programa. Un ejemplo contribuirá a clarificar las cosas:

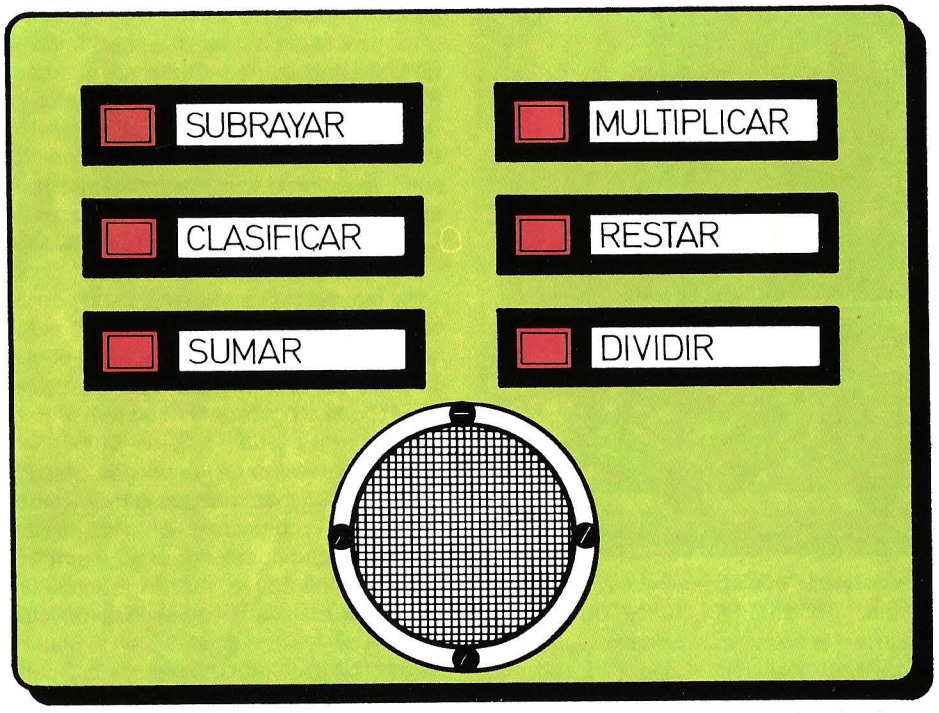
```
PROGRAM operaciones (INPUT,OUTPUT);
VAR A,B,suma,producto:REAL;
PROCEDURE sumar (num1,num2:REAL;
VAR resulsuma:REAL);
BEGIN
        resulsuma:=num1+num2
END;
PROCEDURE multiplicar (num3,num4:
  REAL;VAR resulproducto:REAL);
BEGIN
        resulproducto:=num3*num4
END;
BEGIN (* DEL PROGRAMA PRINCIPAL *)
        READLN(A,B);
        sumar(A,B,suma);
        multiplicar(A,B,producto);
        WRITELN('SUMA=,suma);
        WRITELN('PRODUCTO=',producto)
END (* del programa *).
```

En el programa intervienen dos procedimientos con parámetros, denominados "sumar" y "multiplicar". En ellos, los parámetros formales son: "num1", "num2", "num3" y "num4" (sólo de entrada) y "resulsuma" y "resulproducto" (de salida).
La llamada a los procedimientos con parámetros se realiza en la forma habitual, por medio de su identificador; si bien, en este caso, hay que añadir tras él una lista de parámetros encerrada entre paréntesis.
Estos parámetros, incluidos en la sentencia de llamada, se denominan *parámetros actuales,* por ser los que van a asignar valores a los parámetros formales del procedimiento. En el ejemplo, al parámetro formal "numi" se le asignará el valor del parámetro actual "A" en el momento de la llamada. Y análogamente para los restantes parámetros de entrada al procedimiento. El procedimiento operará dichos valores para producir un resultado que se asignará al correspondiente parámetro de

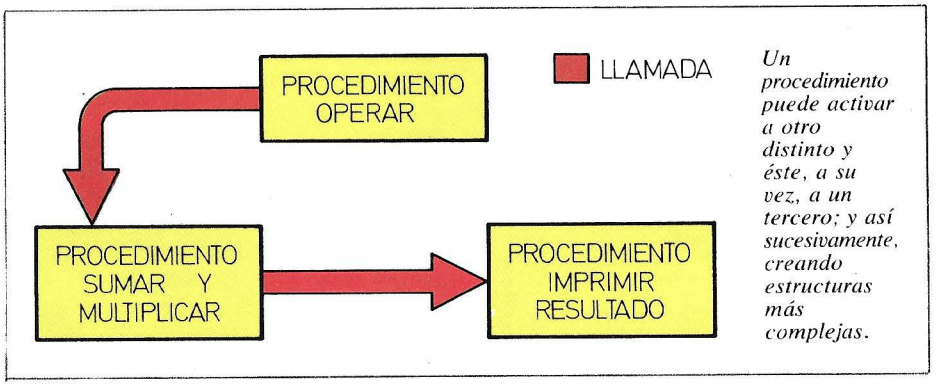
TABLA DE COMANDOS-PASCAL

COMANDO	COMETIDO	OBSERVACIONES
PROCEDURE <id.>; <declaraciones>; bloque de sentencias>;	Declaración de un subprograma o procedimiento sin parámetros.	Zona de declaraciones y definiciones.
PROCEDURE <id.>; (<lista>); <declaraciones>; bloque de sentencias>;	Declaración de un subprograma o procedimiento con parámetros.	Zona de declaraciones y definiciones.
id. {, id. >, ...} : <tipo>;	Parámetros formales de entrada.	Cabecera de la declaración de procedimientos con parámetros.
VAR id. {, id. >, ...} : <tipo>;	Parámetros formales de entrada/salida.	Cabecera de la declaración de procedimientos con parámetros.

NOTA: <id. >: identificador válido PASCAL. <tipo>: tipo de datos. <lista>: lista de parámetros formales de entrada y/o salida de un procedimiento.



La activación de un procedimiento se realiza mediante una simple llamada a su identificador.



Un procedimiento puede activar a otro distinto y éste, a su vez, a un tercero; y así sucesivamente, creando estructuras más complejas.

salida; su valor estará disponible en el parámetro actual de salida de la sentencia de llamada ("suma" o "producto"), una vez concluida la ejecución del procedimiento. Cada parámetro actual inicializa a su correspondiente parámetro formal; por ello, es necesario que éstos se correspondan en número, orden y tipo, en la sentencia de llamada y en la declaración del procedimiento.

Suponga que al ejecutar el programa se introducen para "A" y "B" los valores 3 y 5, respectivamente. En el momento de llamar al procedimiento "sumar", el valor "A" se transfiere a "num1" y el de "B" a "num2". El procedimiento asignará entonces el valor 8 a "resulsuma", y al devolver el control a la sentencia de llamada, este valor será transferido a "suma", con lo que este resultado estará disponible en dicho parámetro actual para imprimirlo en la pantalla o en el periférico de salida.

Si un parámetro formal es sólo de entrada, el correspondiente parámetro actual puede ser una constante, una expresión, una variable, o incluso el identificador de llamada a otro procedimiento. No obstante, si el parámetro formal es también de salida, su parámetro actual debe ser obligatoriamente una variable. Por ejemplo:

```
sumar(3,5+B,suma);
multiplicar(3,5,9);
```

Así la llamada anterior al procedimiento "sumar" del programa "operaciones", es correcta, pero no lo es la llamada a "multiplicar", puesto que el parámetro actual de salida es una constante (9) en lugar de ser una variable.

MS/DOS (8)

Ficheros de comandos

Cualquier usuario de un ordenador se ha visto obligado a armarse de paciencia y aguardar expectante la conclusión de un proceso, para proseguir su actividad con la máquina de acuerdo al resultado obtenido. El MS/DOS resuelve estas ingratas esperas, a la vez que pone a disposición del usuario una herramienta que le permite "personalizar" el sistema operativo. La clave se encuentra en el uso de ficheros cuyo contenido son comandos, expresados con el mismo formato que el que tienen cuando son introducidos a través del teclado. Dichos ficheros reciben el nombre de *ficheros de bloques de comandos* o "batch files".

GENERALIDADES

Los ficheros de bloques de comandos constituyen una de las características más relevantes del sistema operativo MS/DOS. Estos ficheros deben incluir obligatoriamente como nombre de extensión el sufijo "BAT", para que sean reconocidos como ficheros de comandos por el sistema operativo. Su contenido —un comando por cada registro—, puede ser creado mediante un editor de textos, como el EDLIN, o con el auxilio de un procesador de textos, siempre y cuando dicho procesador produzca ficheros de texto ASCII sin ningún carácter de control especial.

La ejecución de un fichero de bloques de comandos se produce al introducir por el teclado el nombre del referido fichero; no es necesario en este caso especificar el nombre de la extensión. En consecuencia, el sistema operativo buscará primero entre los comandos internos, luego entre los externos, a continuación entre los ficheros cuya extensión sea COM y EXE y, finalmente, si el nombre especificado no se refiere a ninguno de los tipos anteriores, buscará entre los ficheros cuya extensión sea BAT. Una vez localizado el fichero, da comienzo la ejecución del comando o programa indicado en el primer registro del mismo. Tras procesarlo, se pasa a la ejecución del próximo comando, y así sucesivamente hasta alcanzar el final del fichero. Hay que tener en cuenta que el MS/DOS lleva la cuenta de los comandos ejecutados y siempre conoce cuál es el próximo comando a ejecutar.

APLICACIONES

El uso de los ficheros de bloques de comandos permite al usuario crear un entorno de comandos a su medida. De esta forma, las operaciones más frecuentes o las que sigan siempre una misma metodología, son ejecutadas con el mínimo esfuerzo.

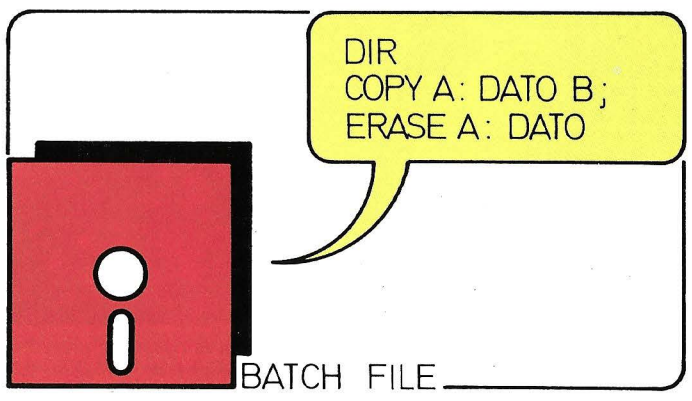
Una de las aplicaciones más evidentes, aunque no por ello menos interesante, es el cambio de nombre de los comandos, ya sea para adaptarlos al gusto del usuario, o bien para construir abreviaturas y nombres nemotécnicos que ayuden a simplificar su empleo.

Por ejemplo, se puede crear un archivo de nombre DISCO.BAT, cuyo contenido coincida con los siguientes comandos:

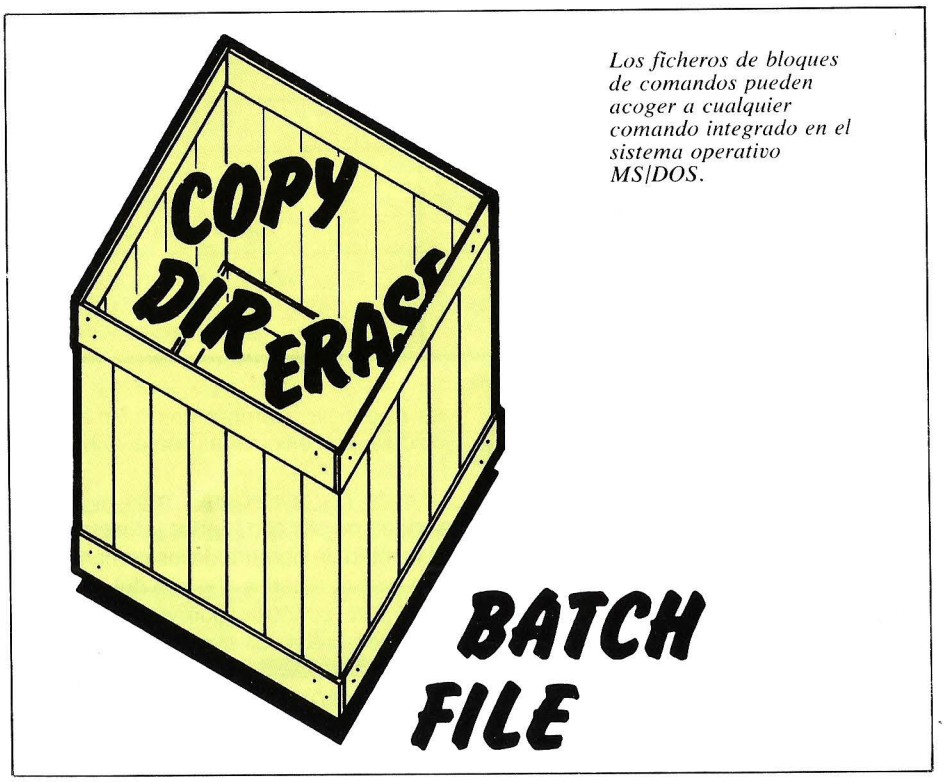
```
DIR        (Lista los ficheros del disco)
CHKDSK (Informa del estado del disco).
```

Así pues, cuando se teclee la palabra DISCO y se pulse la tecla ENTER, el resultado será una información completa del contenido y estado del disco. Las ventajas son indudables; sin ir más lejos, no hace falta recordar los nombres y la formulación de la serie de comandos necesarios para acometer una determinada tarea.

El uso de ficheros de bloques de comandos que encierren llamadas a otros ficheros de ese mismo tipo, exige adoptar ciertas precauciones. Hay que recordar que los ficheros de bloques de comandos



Mediante los ficheros "batch" o ficheros de bloques de comandos, se instruye al ordenador para que ejecute, encadenadamente y de forma automática, una serie de comandos.

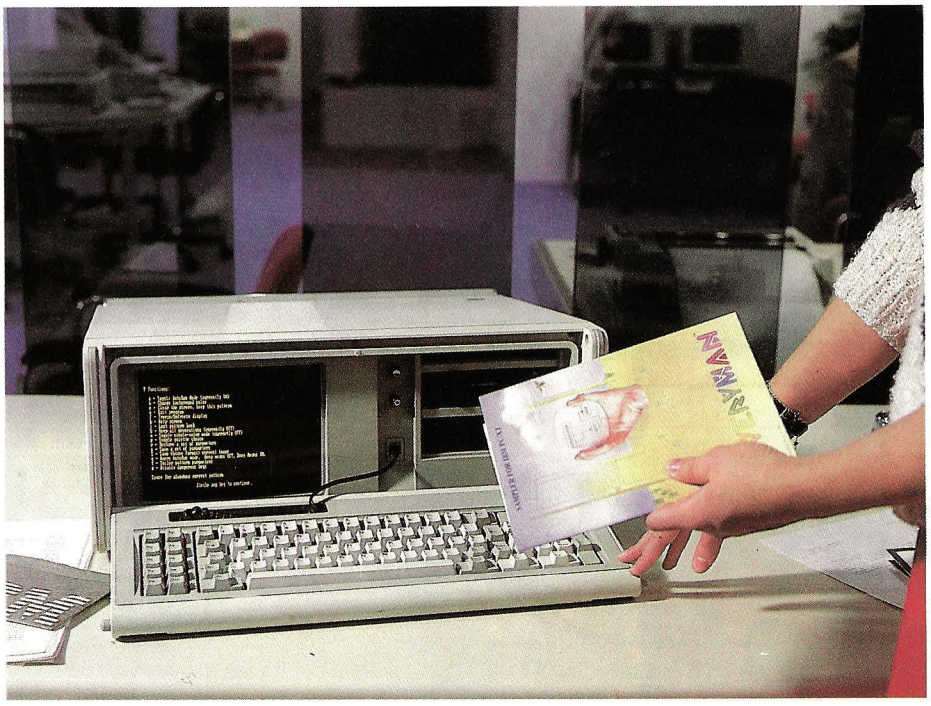


Los ficheros de bloques de comandos pueden acoger a cualquier comando integrado en el sistema operativo MS/DOS.

pueden ir encadenados, pero nunca anidados. En otras palabras: suponga que un fichero de bloques de comandos, A, tiene en uno de sus registros el nombre de otro fichero de bloques de comandos, B; al terminar la ejecución del fichero B, el control no volverá al fichero A, sino que se interrumpirá la ejecución de dicho fichero en el registro siguiente al que hacía referencia al fichero B.

Las aplicaciones que requieren una serie de pasos englobados dentro de una única unidad funcional, constituyen ejemplos típicos en los que resulta adecuado el uso de ficheros de bloques de comandos. En ellas, el fichero de bloques de comandos resumirá los diversos pasos a realizar, ordenados secuencialmente, de tal forma que cuando termine uno de ellos empezará la ejecución del siguiente. Como ejemplo, cabe citar la confección y puesta a punto de un programa en PASCAL, el cual requiere los siguientes pasos: a) creación del fichero fuente en PASCAL, b) compilación, c) edición de enlaces y d) ejecución; cada uno de estos pasos estará en correspondencia con un registro del fichero de bloques de comandos.

Dada la gran variedad de casos tratables y las distintas necesidades de cada usuario, hay que concluir resaltando la flexibilidad, seguridad y velocidad que aporta este método, dejando el campo abierto a la iniciativa del usuario.



El uso de los denominados "ficheros de bloques de comandos" libera la constante atención del usuario sobre el ordenador. Un fichero de esta categoría puede contener toda una secuencia de comandos, cuya ejecución encadenada se producirá de forma automática con sólo llamar al fichero en cuestión.

USO DE PARAMETROS

El uso práctico de los ficheros de bloques de comandos se flexibiliza en gran medida con el empleo de parámetros asociados. Estos permitirán que un mismo fichero de bloques de comandos sirva para distintas tareas, según sea el parámetro indicado.

Los parámetros se colocan a continuación del nombre del fichero y sustituyen dentro de éste a los símbolos %1, %2, % 3... de acuerdo al lugar que ocupan los diversos parámetros en la línea de comando. Por ejemplo, si se desea construir un comando que copie todos los ficheros de un disco a otro, permitiendo la definición de los discos de origen y destino, puede crearse un fichero de bloques de comandos de nombre TRANS.BAT. Su cometido coincidirá con:

```
COPY 1:*.*2:
```

Para activar su funcionamiento será necesario incluir dos parámetros junto al nombre. Así, por ejemplo, TRANS A B particu-

larizará el contenido del fichero de comandos de la siguiente forma:

COPY A:*.*B:

Ordenando la copia de todos los ficheros del disco A en el B.
La presencia de los parámetros permite elegir a voluntad los discos afectados, con la única limitación de que el número de parámetros no puede ser superior a nueve.

COMANDOS PARA FICHEROS DE BLOQUES DE COMANDOS

El contenido de un fichero de bloques de comandos no se limita exclusivamente a comandos del sistema, sino que también admite comandos que gobiernen la ejecución de estos ficheros. Las revisiones del DOS anteriores a la 2.00 sólo admiten los comandos REM y PAUSE; no obstante, a partir de la revisión 2.00 son aceptados todos los comandos que se describen a continuación:

- **REM**

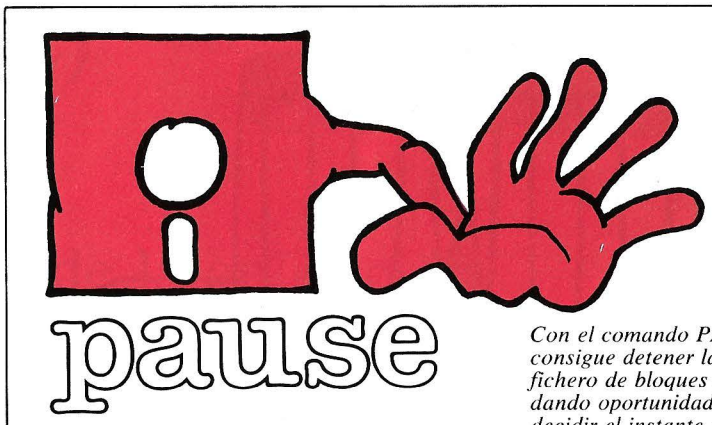
No tiene ningún efecto específico. Se utiliza únicamente para introducir comentarios que ayuden a interpretar la función de los comandos.

- **PAUSE**

Detiene la ejecución de un fichero de comandos; ésta no se reanuda hasta que se accione alguna tecla. La pausa puede ser de utilidad, por ejemplo, cuando está saliendo una gran cantidad de información por la pantalla y no hay tiempo material para examinarla; su presencia detendrá el flujo de información temporalmente.
A menudo no se desea que la ejecución de un programa sea enteramente automática, por motivos de seguridad; algún acontecimiento imprevisto puede llevar a resultados catastróficos. Aquí también es de gran utilidad la inclusión de una pausa en los lugares críticos para poder controlar si se desea o no la realización del paso siguiente.

- **ECHO**

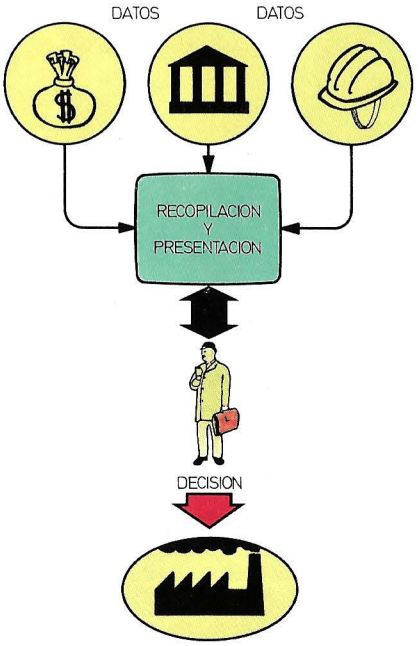
Este comando tiene dos estados: ON y OFF, activado y desactivado. Cuando está activo, el efecto que produce es la impresión por pantalla de los comandos que van siendo ejecutados; en caso contrario, no produce efectos visibles. La elección de uno u otro estado depende del objetivo que se persiga. En el caso de que se esté desarrollando una aplicación, interesará

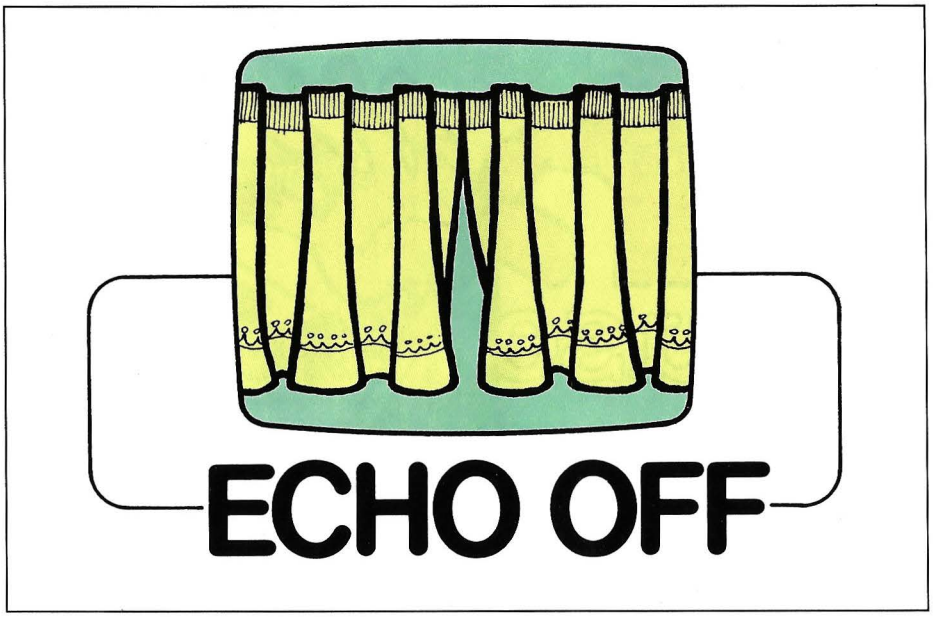


Con el comando PAUSE se consigue detener la ejecución del fichero de bloques de comandos, dando oportunidad al usuario para decidir el instante en que debe reanudarse el proceso.

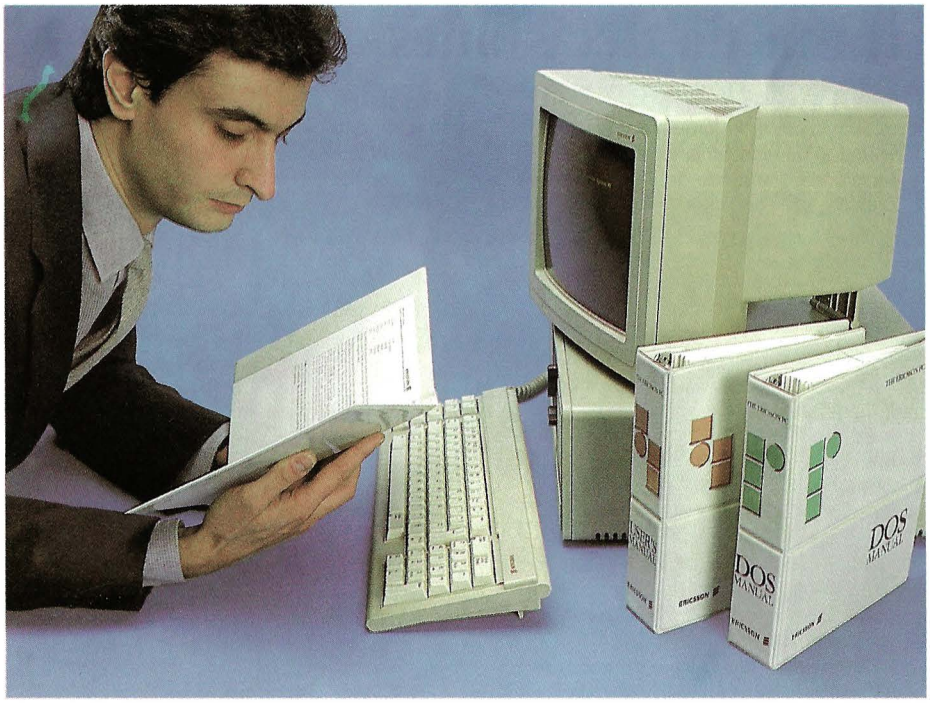
El ordenador como ayuda en la toma de decisiones

Una de las aplicaciones más espectaculares del ordenador es la de ayudar al usuario en la toma de decisiones. Partiendo de un conjunto de datos, y aplicando criterios previamente definidos y plasmados en un programa, el ordenador elabora y presenta los datos bajo distintas perspectivas y muestra las alternativas a que conducen la modificación de ciertos datos. Son múltiples los ejemplos cotidianos que pueden ilustrar este ámbito de aplicación del ordenador. Por ejemplo, en el entorno empresarial, es posible, a partir de los diferentes costes pormenorizados de una cadena de producción, analizar las variaciones en el precio final de los productos; variaciones resultantes de cualquier fluctuación en los elementos que determinan el coste en la etapa de producción. Y todo ello en un breve espacio de tiempo.
Otro ejemplo, algo más extremo, cabe encontrarlo en los métodos para la defensa del espacio aéreo de un país. En este caso, partiendo de los datos suministrados por elementos como el radar y observadores humanos, el ordenador analiza la situación que se crea. Para ello, asigna prioridades a los diversos aparatos enemigos en función de su distancia a puntos neurálgicos del país y de la velocidad con la que se acercan a los mismos. Finalmente, presenta al responsable un cuadro de la situación, de manera que este último pueda decidir la zona en la que debe concentrar sus fuerzas. Todo el proceso se realiza en un tiempo mínimo, permitiendo que las decisiones se tomen de manera casi instantánea respecto al desarrollo de los acontecimientos.





La visualización por pantalla de los comandos del fichero "batch" ejecutados, es controlable por medio del comando ECHO. Los parámetros ON y OFF ordenan su activación o desactivación.



La posibilidad de trabajar con ficheros de bloques de comandos, permite al usuario crear un repertorio de órdenes evolucionadas a su medida. De esta forma, las operaciones más frecuentes o ajustadas a una misma metodología, serán ejecutadas con el mínimo esfuerzo.

ver los comandos ejecutados; ello facilitará la depuración de los programas. En otras ocasiones puede resultar molesto o de mal efecto el hecho de que aparezcan los comandos ejecutados.

Un uso especial de este comando se produce al emplearlo para imprimir mensajes en la pantalla con el formato:

ECHO mensaje

El mensaje siempre aparece, aunque el estado de ECHO sea OFF.

- **SHIFT**

Interviene en los casos en los que se manejan parámetros. Su efecto es lograr que todos los parámetros se desplacen un lugar hacia la izquierda, desapareciendo el primero de ellos. Esto es: el símbolo %1 toma el valor que tenía %2, %2 toma el valor de %3 y así sucesivamente.

- **GOTO etiqueta**

El efecto de este comando es variar el flujo secuencial de los comandos, dando un salto dentro del archivo de bloques de comandos hacia el lugar indicado por la etiqueta. La etiqueta se distingue por estar precedida por el signo dos puntos. Suele asociarse a sentencias condicionales. Por ejemplo:

```
:OTRO
ERASE A: %1
SHIFT
GOTO OTRO
```

Su efecto será borrar todos los ficheros indicados en los parámetros.

- **IF**

Este comando comprueba una condición lógica; si es cierta ejecuta el comando situado a continuación de IF, y en caso contrario lo ignora. El formato es:

IF Condición Comando

Algunas de las condiciones lógicas que se pueden comprobar son la existencia de un fichero, la ocurrencia de un error, la comparación entre dos cadenas de caracteres...

- **FOR**

Se emplea para ordenar la ejecución repetitiva de un determinado comando, mediante una variable simbólica que va precedida por dos signos de "porcentaje" (%%). Esta variable adopta un valor de una lista de posibles valores. El formato es el siguiente:

FOR %%Nombre IN (Lista de valores)
DO VOL %%Nombre:

Por ejemplo:

FOR %%Nombre IN (A, B, C)
DO VOL %%Nombre:

alude a la etiqueta del volumen de los discos A, B y C.

Multitexto (2)

Menús y opciones de la aplicación

Continuando con el análisis del paquete MULTITEXTO, vamos a detallar las pantallas y submenús que visualizará el usuario después de haber optado por alguno de los puntos del menú principal. Tras dicho análisis, se describirán las principales funciones y utilidades aportadas por el programa. En el próximo capítulo concluirá el estudio con una típica sesión de trabajo; en ella se hará especial hincapié en las tareas más frecuentes en el ámbito del proceso de textos con el programa MULTITEXTO.

PANTALLAS Y MENUS

A partir del menú principal, el usuario puede activar cualquiera de las nueve opciones que en él se le ofrecen; el programa continuará presentando una serie de pantallas y menús auxiliares cuyo cometido se detalla a continuación:

1. *Editar un documento antiguo*
Esta es la primera opción del menú principal. Al seleccionarla aparece en la pantalla un nuevo menú que permite al usuario especificar los siguientes datos:
— El drive (unidad de disco) que contiene el documento a editar.
— El nombre de dicho documento, el cual no es obligatorio especificar completo: con sólo los 8 primeros caracteres, el programa tendrá suficiente información para localizarlo.
Como información suplementaria, el MULTITEXTO indicará los caracteres y páginas que quedan disponibles en el drive en que se encuentra almacenado, y una lista alfabética con todos los nombres de los documentos archivados.
A continuación, sin más que pulsar la tecla "RETURN", aparecerá una nueva pantalla en la que se resumen las características del documento seleccionado en la anterior. Entre otros datos, la pantalla muestra: el número total de páginas, nombre del documento, autor, palabras clave que contiene, comentarios, fecha de creación y fecha de la última modificación.

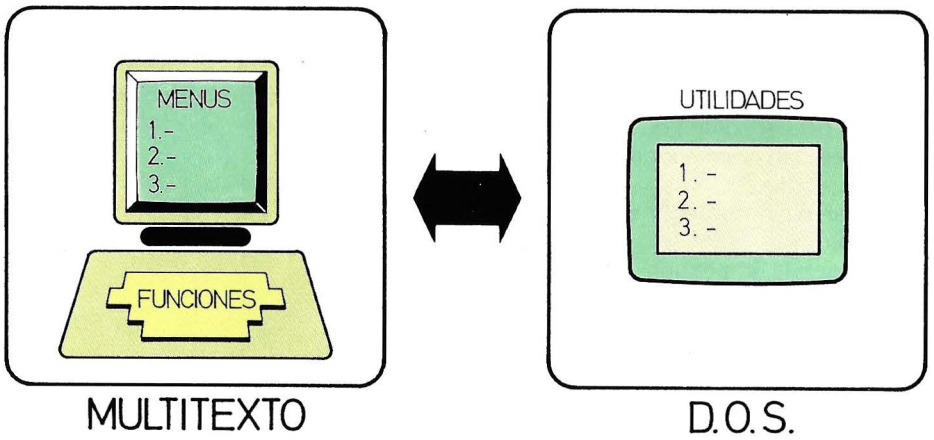
2. *Crear un documento nuevo*
Si la opción elegida en el menú principal es la que permite crear nuevos documentos, el programa MULTITEXTO se comportará de forma análoga a la de edición de documentos antiguos. En la pantalla aparecerán sucesivamente dos pantallas: una para indicar el nombre del nuevo fichero y otra para resumir sus características.

3. *Imprimir un documento*
Después de seleccionar la tercera opción del menú principal, aparecerá una pantalla denominada "Parámetros para imprimir un documento". En ella hay que especificar, entre otros, los siguientes parámetros: primera y última página que se desea imprimir, márgenes izquierdo y superior, tipo de impresora en la que se escribirá el texto, justificación del margen derecho, tamaño de la impresión (caracteres por pulgada), opción para imprimir resumen del documento, fecha y hora en que se desea producir el informe.

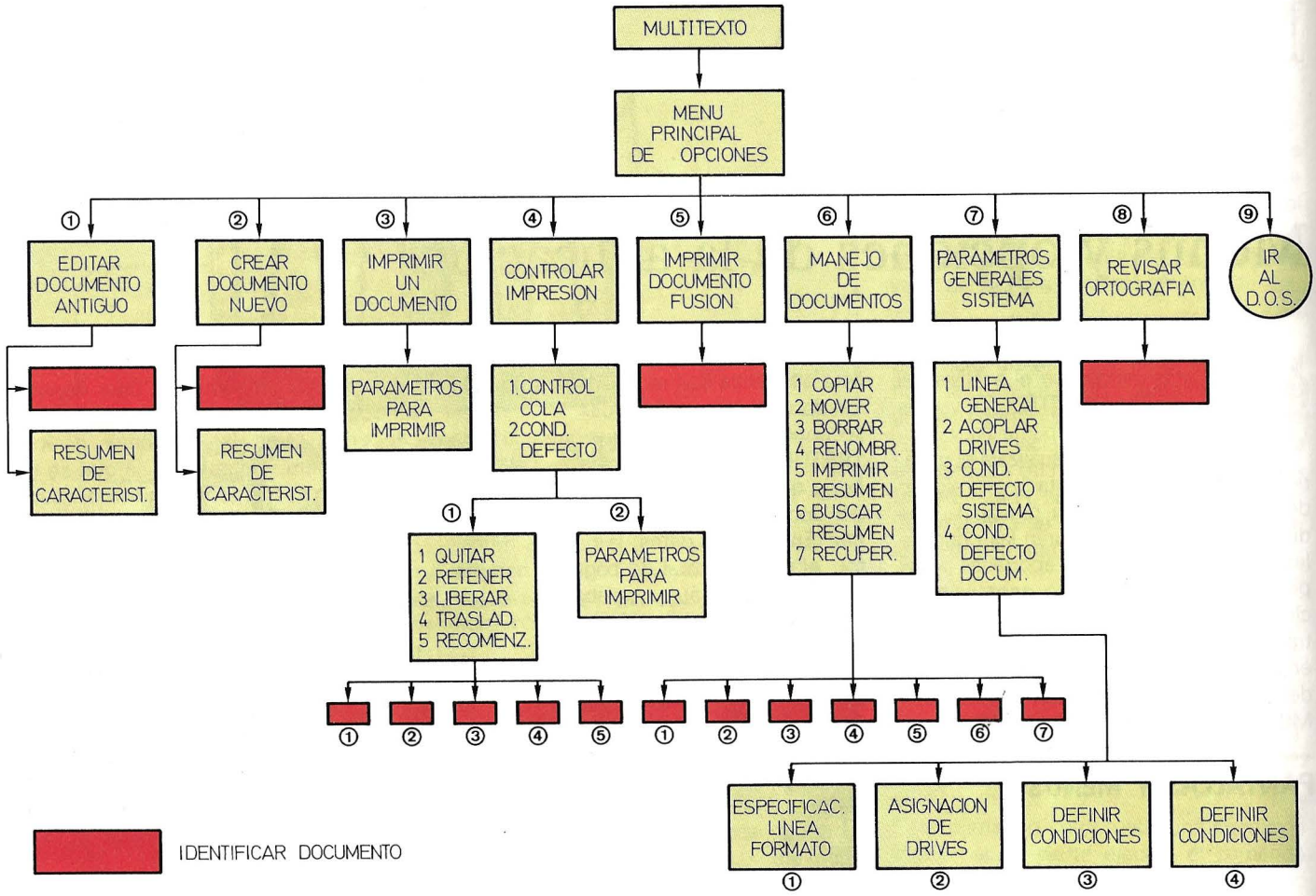
4. *Opciones para controlar la impresión*
A través de esta opción, la aplicación MULTITEXTO permitirá al usuario acceder a la cola de impresión; para ello le ofrecerá un nuevo menú en el que se abren dos posibilidades:

a—Control de la cola de impresión.
b—Condiciones por defecto para la impresión.

La primera se utilizará para gestionar los documentos que se han ordenado imprimir desde la opción 3 del menú principal,



La explotación del paquete MULTITEXTO supone la intervención de dos entes propios: una estructura arborescente de menús (visible en la pantalla) y un repertorio de funciones accionadas por medio de las oportunas combinaciones de teclas. A ello se añade un ente propio del MS/DOS (PC/DOS), cual es el conjunto de utilidades del sistema operativo.



Estructura arborescente de la aplicación MULTITEXTO. Sus múltiples opciones derivan de las nueve alternativas básicas que ofrece el menú principal.

mientras que la segunda permite especificar o modificar los parámetros asignados en la opción de impresión. Respectivamente, utilizan un menú y una pantalla. En el menú para el control de la cola de impresión existen cinco opciones distintas:

a—Retirar un documento de la cola de impresión.
b—Retener la impresión de un documento.
c—Liberar la impresión de un documento.
d—Trasladar un documento al principio de la cola.

MULTITEXTO
Versión 3.30

1) Editar un Documento Antiguo
2) Crear un Documento Nuevo

3) Imprimir un Documento
4) Opciones para Controlar la Impresión
5) Imprimir un Documento Fusión

6) Opciones para el Manejo de Documentos
7) Prefijar Parámetros Generales del Sistema
8) Revisar la Ortografía de un Documento
9) Volver al Sistema Operativo D.O.S.

FUNCION DESEADA:

Introduzca el número de la función y pulse (RETURN)
Pulse (Shift) (F1) para llamar al Menú de Ayudas

S: ↓ N: ↓

Menú principal que da entrada a las opciones básicas del paquete MULTITEXTO.

e—Recomenzar la impresión del documento actual.

A su vez, en la pantalla para la especificación de condiciones por defecto para la impresión, se pueden introducir los mismos datos descritos anteriormente en el caso de la pantalla denominada "Parámetros para imprimir un documento". La aparición de esta pantalla en dos puntos distintos de la estructura de menús del MULTITEXTO puede parecer redundante, aunque ello no es así: su uso a partir de la opción 3 del menú principal implica la definición de parámetros para un único documento, mientras que su empleo desde la opción 4 del menú principal permite realizar una definición general de parámetros que afectará a todos los documentos.

5. *Imprimir un documento fusión*
Desde la quinta opción del menú principal se pueden combinar dos documentos, a los que denominaremos primario y secundario, sintetizándolos en un documento único. Tras seleccionar la opción 5, aparecerá una pantalla denominada "Fusión de documentos"; en ella hay que indicar, por un lado la denominación y el drive del documento primario, y por otro lado los mismos datos relativos al documento secundario.
Dentro de esta pantalla, el programa también mostrará el espacio aproximado (en caracteres y páginas) disponible en las unidades de disco, y una relación de los documentos archivados en el drive del documento primario. En todo caso, el usuario puede ordenar que aparezca en esta misma pantalla una relación de los documentos incluidos en el drive que aloja al documento secundario.

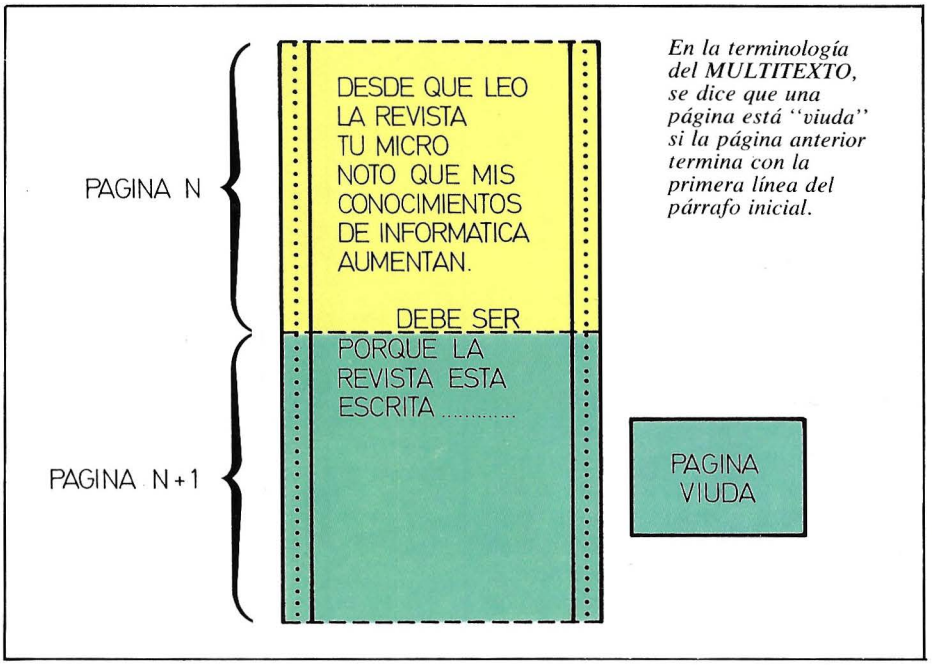
6. *Opciones para el manejo de documentos*
A partir de la opción 6 del menú principal, el usuario puede realizar prácticamente cualquier operación que desee con sus documentos. Las posibilidades aparecen en forma de menú;

a—Copiar un documento.
b—Trasladar un documento.
c—Borrar un documento.
d—Renombrar un documento.
e—Imprimir las pantallas resumen de documentos.
f— Buscar las pantallas resumen de documentos.
g—Recuperar un documento.

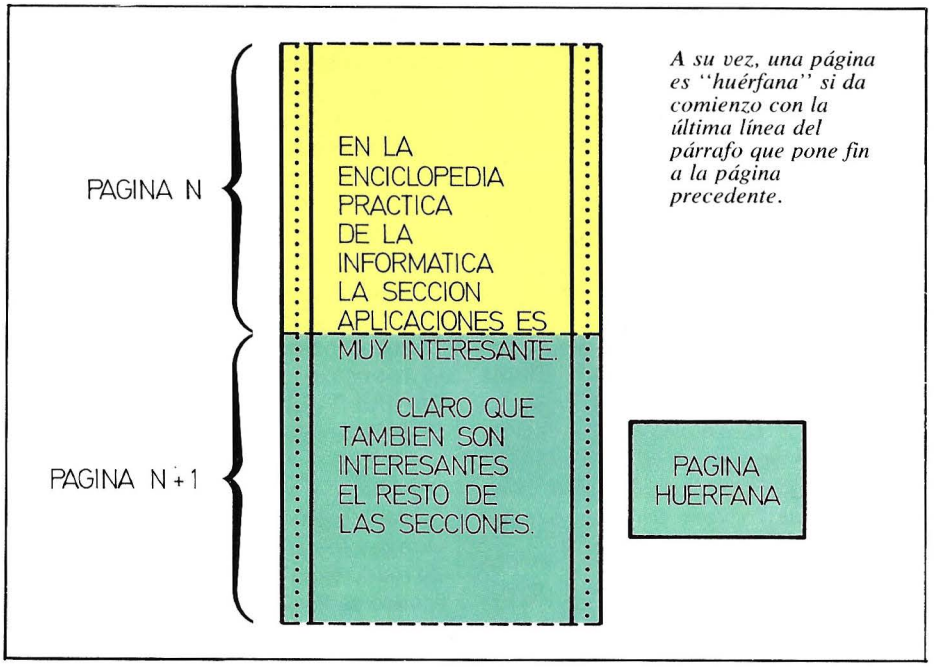
Todas las alternativas son portadoras de pantallas auxiliares en las que el usuario puede especificar las características de la manipulación que desea realizar. De esta forma, en la primera de ellas se indicará el nombre y drive del documento que desea copiar, y el nombre y drive de destino para la copia producida; en la segunda se especificarán los mismos datos, teniendo en cuenta que ahora, en lugar de copiar, se "moverá" la información...

7. *Prefijar parámetros generales del sistema*
Dentro de esta opción, el usuario puede elegir hasta cuatro alternativas de carácter general:

a—Preparar la línea general de formato.
b—Acoplar el sistema con drives instalados.
c—Prefijar las condiciones por defecto del sistema.



En la terminología del MULTITEXTO, se dice que una página está "viuda" si la página anterior termina con la primera línea del párrafo inicial.



A su vez, una página es "huérfana" si da comienzo con la última línea del párrafo que pone fin a la página precedente.

FUNCIONES DEL MULTITEXTO

TECLAS	FUNCION	TECLAS	FUNCION
F10	Archivo página saliendo	Varias	Línea de formato
SHIFT/F10	Archivo página sin salir	Varias	Línea de información
SHIFT/F1	Ayuda	ALT/F2	Longitud de página
F5	Biblioteca	ALT/F1	Marca parpadeante
DEL	Borrado Texto	Menú	Margen derecho
←	Borrado carácter	F7	Mover texto
F6/F9	Búsqueda línea	SHIFT/3	Numeración Automática
F6	Búsqueda palabra	ALT/4	Opciones
ALT/H	Cabecera de página	Menús	Pantalla resumen
F3	Centrado	Varias	Parar
SHIFT/F3	Columna	ALT/F	Pie de página
SHIFT/F2	Combinar páginas	Varias	Procedimientos clave
F8	Copia	CTRL/F2	Prepaginado
SHIFT/F8	Copia externa	←	Retroceso
		↵	Return
ALT/F10	Corrección ortográfica	Menú	Revisión ortográfica
Menú	Crear un documento nuevo	F2	Salto de página
ALT/F3	Cursor 1.er carácter línea	Varias	Sangrado
ALT/F4	Cursor último carácter	Varias	Símbolos
↑	Cursor hacia arriba	Varias	Sobreiluminado
↓	Cursor hacia abajo	Reescribir	Sobreimpresión
→	Cursor a la derecha	ALT/W	Subíndice
←	cursor a la izquierda	Varias	Subrayado
ALT/G	Diferenciar mayúsculas	Varias	Suma
Menú	Editar doc. antiguo	ALT/Q	Superíndice
ESC	Escape	SHIFT/F6	Sustitución
ALT/S	Espacio de unión	Varias	Tabulador
Varias	Fusión	ALT/O	Tachado
SHIFT/F7	Guión condicional	ALT/C	Tamaño de impresión
Varias	Impresión	Varios	Tipos de impresión
INS	Inserción		
Varias	Ir a...		
ALT/Y	Limpiar marcas parpadeantes		

d—Preparar las condiciones por defecto de documentos.

Mediante la primera posibilidad se puede decidir el formato de la línea general que aparecerá en la parte superior de la primera página de cada documento. Con la segunda opción se especificarán todos los drives instalados, indicando en qué unidades de disco se encuentra el sistema, los documentos, la biblioteca y el diccionario. La tercera opción permite definir las condiciones generales de actuación (salto de página automático, etc.). Por último, la cuarta opción se emplea para alterar las condiciones de un documento después de haberlo creado y, por lo tanto, tiene asociadas las mismas pantallas que las opciones 1 y 2 del menú principal.

8. *Revisar la ortografía de un documento*
Al seleccionar la octava opción del menú principal, aparecerá una nueva pantalla por medio de la que el MULTITEXTO solicitará el nombre del documento a revisar y el drive en el que se encuentra. Mientras el sistema esté ocupado en la revisión, aparecerá en la pantalla el mensaje: "OPERACION EN PROCESO" y entre paréntesis se podrán observar el número de palabras mal ortografiadas y el total de palabras ya revisadas; ambos datos irán cambiando mientras el sistema revisa el texto. Cuando finalice dicha revisión, aparecerá el mensaje: "OPERACION COMPLETA—PULSAR CUALQUIER TECLA PARA CONTINUAR".

9. *Volver al sistema operativo DOS*
Evidentemente, el objeto de la novena opción del menú principal no implica el uso de ninguna pantalla o menú; su único cometido es finalizar la sesión de trabajo con el programa MULTITEXTO.

FUNCIONES DEL PROGRAMA MULTITEXTO

En el segundo volumen de la documentación que acompaña al programa en estudio, se encuentra una relación alfabética de las funciones que el usuario puede desencadenar directamente, pulsando una, dos o tres teclas. Mediante estas combinaciones de teclas es posible acceder a 62 funciones distintas que, obviamente, no vamos a detallar dado lo dilatado del repertorio. No obstante, en la tabla adjunta se puede observar una escueta relación de las 62 funciones, junto a la combinación de teclas que permite desencadenar su funcionamiento.

UTILIDADES DEL PROGRAMA MULTITEXTO

Directamente desde el sistema operativo DOS (con el disquete de utilidades en activo), se puede acceder a cinco opciones que aportan una nueva facilidad para el manejo del programa. Estas utilidades aparecen como opciones de un menú principal:

1.—Editar las tablas de impresoras.
2.—Utilidades de procedimientos.
3.—Conversión de ficheros.
4.—Condiciones por defecto de pantalla.
5.—Recuperación de documentos.

Para que sea posible emplear estas utilidades es preciso que el ordenador se encuentre bajo el control del sistema operativo. Por lo tanto, si se está en una sesión de trabajo con el MULTITEXTO, hay que volver al sistema operativo DOS mediante la opción 9 del menú principal. Tras ello, se debe invocar al comando UTIL y pulsar la tecla RETURN, con lo que aparecerá una pantalla con las cinco utilidades antes señaladas. A continuación, se colocará el cursor sobre la utilidad deseada y se accionará de nuevo la tecla RETURN. De esta forma se desencadenará la ejecución de cualquiera de las cinco utilidades.

Trabajando con cadenas

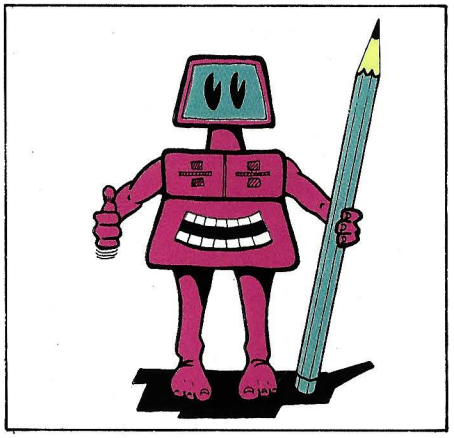
Funciones evolucionadas para la manipulación de cadenas de caracteres

El BASIC es un lenguaje que ofrece grandes posibilidades para el tratamiento de la información alfanumérica. Frente a otros lenguajes de propósito más específico en uno u otro ámbito de la actividad informática, el BASIC ha rechazado la especialización y se emplea en un amplio abanico de aplicaciones. Desde las más triviales, como por ejemplo los juegos, hasta las más serias, como puedan ser las aplicaciones de gestión y enseñanza. En muchas de estas aplicaciones, la facilidad para el manejo de cadenas de caracteres es una de las facetas que hacen del BASIC un lenguaje insustituible y de amplia aplicación.

En este capítulo se estudiarán algunas de las herramientas que ofrece el lenguaje BASIC para la manipulación de cadenas de caracteres. En el apartado de funciones, hay que hablar de las encargadas de la conversión de cadenas a números y de números a cadenas; ampliamente utilizadas y que se encuentran en casi todos los dialectos del BASIC. Las restantes funciones a estudiar son específicas de ciertos dialectos más potentes. Estas últimas resultan de gran utilidad al facilitar el tratamiento de cadenas, aunque no son realmente imprescindibles.

DE NUMEROS A CADENAS

En ocasiones es preciso transformar cadenas de caracteres en números y viceversa. Esta transformación se hace necesaria para permitir el tratamiento de determinados datos numéricos. Con ello se consigue, por ejemplo, la extracción de determinadas cifras aisladas de una forma cómoda y eficaz. La representación de números como cadenas de caracteres ayuda

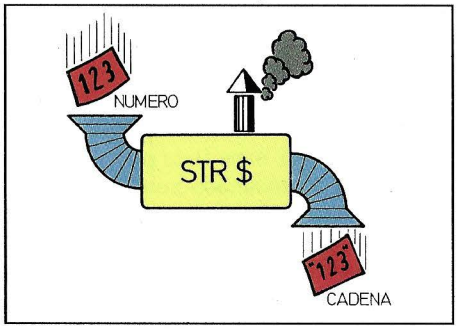


La capacidad del BASIC para la manipulación de cadenas de caracteres, convierte al ordenador en una potente herramienta para el tratamiento de información alfanumérica.

a almacenarlos en archivos y contribuye a economizar memoria en el ordenador.

Consideremos en primer lugar la operación de convertir un dato numérico en una cadena de caracteres. Para ello es necesario emplear la función STR$. Esta se encargará de realizar la transformación adecuada para conseguir dicho resultado. Su aspecto general es el siguiente:

STR$ (<numero>)



La función STR$ da como resultado una cadena cuyos caracteres coinciden con las cifras del número incluido en su argumento.

En donde <numero> representa a la constante o variable numérica que se desea convertir en cadena de caracteres. En consecuencia, las siguientes son todas ellas formulaciones válidas de la función STR$:

STR$ (128)
STR$ (274)
STR$ (A)
STR$ (N)

La función que nos ocupa actúa sobre un dato numérico, convirtiéndolo en una cadena cuyos caracteres coinciden con las cifras que componen el dato de partida. La diferencia entre uno y otro formato no se manifiesta en sus correspondientes representaciones en pantalla. No obstante, sí cabe señalar que los números aparecen en pantalla precedidos por uno o dos espacios en blanco; espacios que son empleados para la representación del signo y que suelen aparecer siempre, aunque el signo no sea visualizado.

A continuación figura un ejemplo que pone de manifiesto la analogía entre ambas representaciones:

```
10 PRINT "128"
20 PRINT 128
RUN <CR>
128
 128
■
```

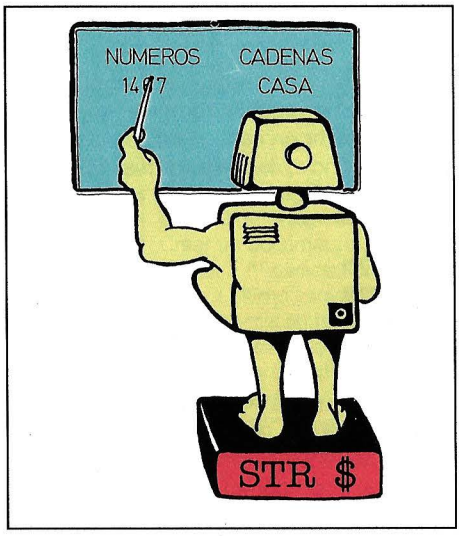
Obsérvese que la segunda presentación del número 128 está desplazada con respecto a la posición inicial de la primera. Ello se debe al motivo anteriormente expuesto.

STR$

Función: Convierte un número en una cadena de caracteres.

Formato: STR$ (<número>)
<número>: número que se desea convertir en cadena.

Ejemplos: STR$(124) STR$(C)
STR$(A) STR$(2458)



es lo mismo realizar una suma de números que una suma de cadenas de caracteres. El resultado de ambas operaciones es bien distinto, como revela el siguiente ejemplo:

```
10 PRINT 128+4
20 PRINT "128"+"4"
RUN <CR>
 132
1284
■
```

En él se observa que la operación "+" realiza una función distinta dependiendo del tipo de datos. Si los datos son numéricos, el resultado será su suma aritmética. Por el contrario, si se trata de cadenas de caracteres, el resultado será su concatenación.

Para hacer aún más evidente este hecho, pueden introducirse las siguientes líneas de programa que ilustrarán tal desplazamiento con mayor claridad:

```
10 PRINT "128";"23";"-234"
20 PRINT 128;23;-234
RUN <CR>
12823-234
 128 23-234
■
```

Los usuarios del ZX-SPECTRUM, notarán que su ordenador no reserva ese espacio para el signo, actuando, en este caso, de forma distinta a la mayor parte de equipos dotados de un traductor BASIC.

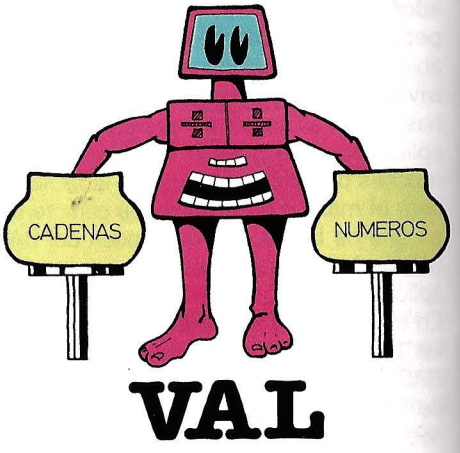
La diferencia entre datos numéricos y alfanuméricos no se limita a una cuestión tan trivial como ésta. Se refleja fundamentalmente en el tipo de operaciones que es posible realizar con ellos. Por ejemplo, no

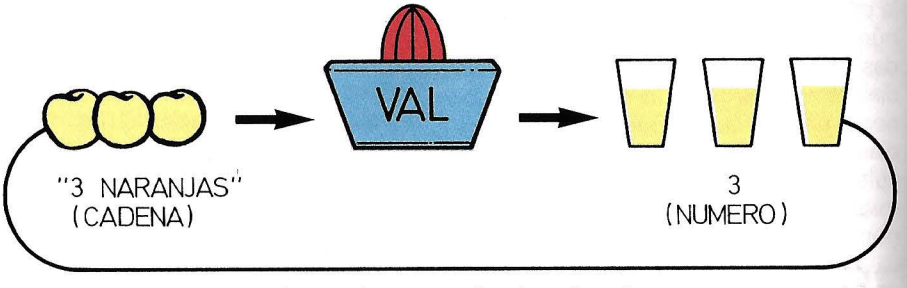
La conversión de datos numéricos en cadenas de caracteres resultará útil, para que sea posible manipular datos numéricos a modo de cadenas de caracteres. En efecto, hay que recordar que existen operaciones fácilmente realizables con cadenas, pero que, sin embargo, son difíciles de realizar con números.

Por ejemplo, para extraer la cifra que en un determinado número corresponde a las centenas, sería necesario recurrir a un programa semejante al que sigue:

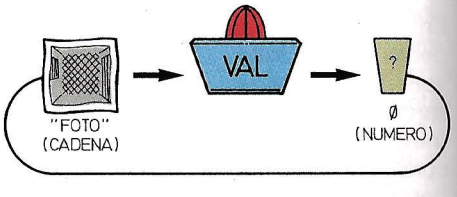
```
 10 A=3457
 20 PRINT "A= ";A
 30 B=INT(A/100)
 40 PRINT "B= ";B
 50 C=INT(A/1000)
 60 PRINT "C= ";C
 70 D=B-C*10
 80 PRINT "D=B-C*10"
 90 PRINT "D= ";D
100 END
```

Comprobemos su eficacia:

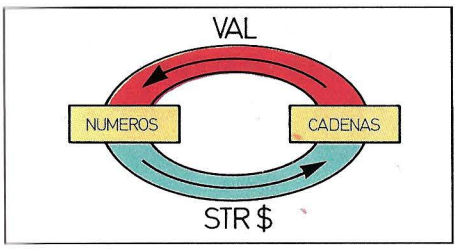




La función VAL obtiene un valor numérico asociado a la cadena de caracteres que constituye su argumento. Concretamente, transforma en dato numérico los primeros caracteres de la cadena que coincidan con cifras.



Si la cadena evaluada no empieza por un carácter numérico, la función VAL genera como resultado el valor cero.



Las funciones STR$ y VAL constituyen un puente que permite la transformación de datos numéricos en alfanuméricos y viceversa.

```
RUN <CR>
A=3457
B=34
C=3
D=B-C*10
D=4
■
```

Esta rutina procede a dividir el dato base (A) por 100 y por 1000 sucesivamente. Con la primera división se coloca la cifra de centenas a extraer (en este caso 4) en la posición de las unidades. La siguiente división obtiene las cifras que quedan a la izquierda de la elegida. Por último, la operación de la línea 70 aisla el dígito buscado, almacenándolo en la variable D.
La misma operación se puede realizar con mayor facilidad e inmediatez si el dato inicial es una cadena de caracteres. La función MID$, comentada en un capítulo anterior, permitirá la extracción del carácter deseado:

```
10 A=3457
20 A$=STR$(A)
30 C$=MID$(A$,LEN(A$)-2,1)
40 PRINT C$
RUN <CR>
4
■
```

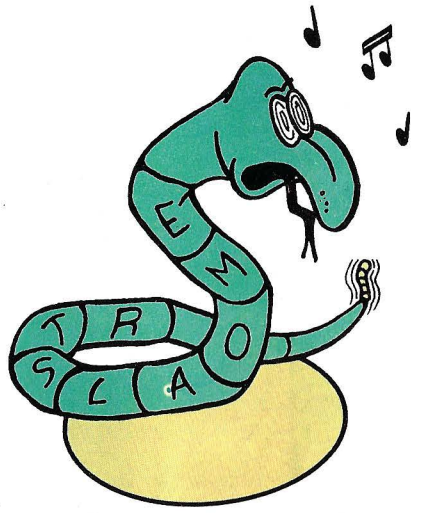
Ahora se consigue la extracción con una sola línea de programa, la 30. Para ello es necesario almacenar el dato inicial en una variable de cadena (línea 20). De esta forma el dato numérico queda transformado en una cadena de caracteres.
Como se ha podido comprobar, la función

VAL

Función: Convierte una cadena en un número.

Formato: VAL (<cadena>)
<cadena>: cadena de caracteres que se desea convertir en un número.

Ejemplos: VAL(A$) VAL(T$)
VAL("128") VAL("2578")



Frente a otros lenguajes de propósito más especializado, el BASIC presenta una gran aptitud para un amplio abanico de aplicaciones. Una de las bazas que apoyan tal versatilidad, reside en su capacidad para el tratamiento de datos alfanuméricos o cadenas de caracteres.

STR$ permite almacenar un número en una variable de cadena. Este paso hace posible realizar algunos tipos de operaciones que resultan difíciles, y a veces imposibles, con números.
También es posible realizar la conversión inversa. Es decir, el paso de cadena de caracteres a dato numérico. Para ello hay que emplear la función VAL cuya especialidad es obtener el valor numérico asociado a una cadena de caracteres. Desde luego, es necesario que la cadena inicial esté constituida por cifras. Su formato es el siguiente:

VAL (<cadena>)

El argumento <cadena> representa al dato alfanumérico que se desea convertir en un número. He aquí algunos ejemplos:

VAL (A$)
VAL ("273")
VAL ("127")
VAL (T$)

Su actuación se reduce a convertir los caracteres de la cadena que correspondan a números en sus respectivos valores numéricos. En el caso de que la cadena no esté compuesta únicamente por cifras, se realizará la conversión sólo hasta el primer carácter no numérico. Si se da la circunstancia de que el primer carácter de la cadena no es un número, el valor devuelto por esta función será cero.
En el BASIC de Sinclair (ZX-SPECTRUM y ZX-81), la función STR$ presenta una singularidad: si al evaluar una cadena de caracteres se encuentran caracteres no numéricos, éstos se tomarán como nombres de variables; se buscará su valor correspondiente y se operará con ellos. En otras palabras, la cadena es interpretada cual si se tratara de una fórmula.

GENERACION DE CADENAS

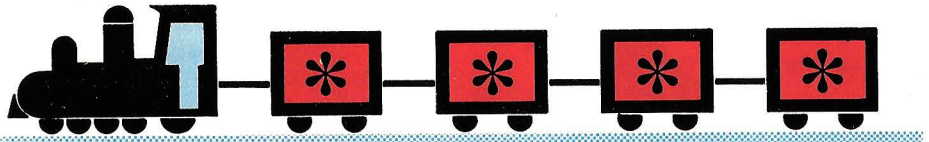
En ocasiones es necesario crear cadenas de caracteres de una determinada longitud y que estén formadas únicamente por espacios en blanco. Quizá el lector se pregunte cuál es la utilidad práctica de semejante tipo de cadena. Suponga, por ejemplo, que es necesario disponer de una cadena de longitud específica. Dicha cadena estará formada por una serie de caracteres que constituyen realmente el dato con el que se ha de trabajar. Nada garantiza que el número de caracteres vaya a ser precisamente el estipulado. Lo más normal es que el espacio destinado a contener la información sea grande, y que sobre sitio para contener la cadena. Por ello, es necesario llenar el espacio que sobra con caracteres en blanco. Esta puede ser una alternativa útil, por ejemplo, a la hora de almacenar información en un archivo o presentarla en pantalla (re-

SPACE$

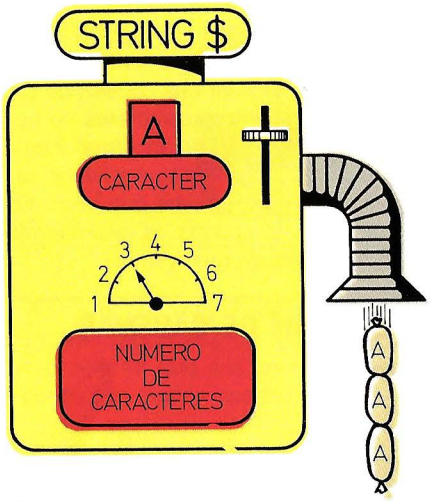
Función: Genera una cadena de caracteres, compuesta por la repetición de espacios en blanco un determinado número de veces.

Formato: SPACE$ (<número>)
<número>: número de espacios en blanco que deben conformar la cadena.

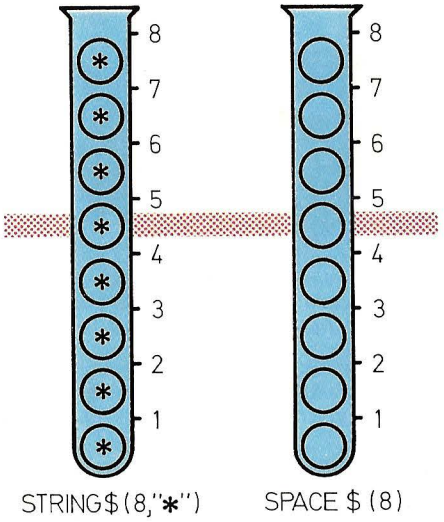
Ejemplos: SPACE$(7) SPACE$(12)
SPACE$(N) SPACE$(S)



STRING$ es una función BASIC que permite construir cadenas, de una determinada longitud, en base a la repetición de un determinado carácter.



La puesta en actividad de la función STRING$ exige dos datos: el carácter a repetir y el número de caracteres que deben constituir la cadena resultante.



STRING$ construye una cadena integrada por la repetición del carácter que indique el programador, mientras que la función SPACE$ sintetiza la cadena resultante a base de espacios en blanco.

STRING$

Función: Genera una cadena de caracteres formada por la repetición, un determinado número de veces, del carácter que se especifique.

Formato: STRING$ (<num1>,<num2>
ó
STRING$ (<num1>,<cadena>)
<num1>: número de caracteres que han de formar la cadena.
<num2>: código ASCII del carácter a utilizar.
<cadena>: el primer carácter de esta cadena será el que constituya la nueva cadena.

Ejemplos: STRING$(7,"CASA") STRING$(N,P)
STRING$(12,56) STRING(N,A$)

cuerde que no todos los ordenadores poseen la socorrida instrucción PRINT AT). Una solución al problema planteado consiste en utilizar la siguiente rutina, que no por simple deja de ser menos interesante y efectiva.

```
 5 INPUT B
10 LET B$=""
20 FOR I=1 TO B
30 LET B$=B$+" "
40 NEXT I
```

La ejecución de de esta rutina crea una cadena B$ formada por un número B de espacios en blanco. Todo su secreto reside en añadir blancos a una cadena vacía. De todas formas, este mismo resultado se puede obtener mediante el empleo de la función SPACE$. Siempre y cuando ésta forme parte del repertorio del BASIC del ordenador utilizado. Su formato es el siguiente:

SPACE$ (<número>)

La zona <número> estará ocupada por un número natural, cuya misión es indicar al ordenador el número de espacios en blanco que van a conformar la cadena.

SPACE$ (6)
SPACE$ (N)

Volviendo al ejemplo anterior, hubiera bastado con la siguiente línea de programa para conseguir el mismo resultado:

LET B$=SPACE$ (B)

La función SPACE$ puede utilizarse en compañía de otras funciones para el tratamiento de cadenas de caracteres. La siguiente rutina hará que cualquier cadena introducida se imprima en pantalla, ocupando diez espacios delimitados por el símbolo "/".

```
100 INPUT B$
110 LET A$=SPACE$ (10-LEN (B$))
120 LET B$=B$+A$
130 PRINT "/";B$;"/"
```

En la línea 110 se hace uso de LEN para calcular los espacios que le faltan a B$ para llegar a los 10 caracteres de longitud. El cálculo constituye el argumento de SPACE$, logrando así una cadena (A$) con el número adecuado de blancos. La línea 120 es la encargada de ensamblar ambas cadenas, para formar una cadena final de 10 caracteres de longitud. Por último, en la línea 130 se imprime la cadena obtenida, flanqueada por los símbolos

Sistemas de numeración

Desde los tiempos más remotos, el hombre ha utilizado sistemas de numeración para cuantificar objetos y acontecimientos. Enumerar y llevar la cuenta de todo tipo de objetos ha sido una necesidad atemporal, necesidad que dio pie al desarrollo de sistemas de numeración que le permitieran conocer sus pertenencias, sus deudas o su edad.

Entre los diversos sistemas inventados, el más extendido es el decimal. La razón es muy simple: poseemos diez dedos en las manos.

Seguramente, el primer contable de la historia utilizó sus dedos. Aún hoy en día estos apéndices son considerados como la "calculadora" más elemental. El principal defecto de este método de cálculo reside en el hecho de que sólo se puede llegar hasta diez. Para cifras mayores es necesario tomar una referencia externa.

Suponga que se desea contar una serie de objetos. La cuenta empieza extendiendo un dedo por cada objeto contabilizado. Al llegar al décimo objeto se terminan los dedos.

Este problema puede solventarse buscando a alguien que cuente las unidades de diez dedos acumuladas. Al final de la cuenta se verifican las decenas y los dedos sueltos, con lo que se obtiene la cantidad total.

Cuando el segundo contador termina sus dedos, es necesario llamar a un tercero. Este último contará las centenas o decenas de decenas. Y así sucesivamente.

Este es, justamente, el método que se utiliza para escribir cifras. La cantidad 23 indica 2 decenas y 3 "dedos sueltos" o unidades. Cuando se acumulan más de 9 decenas se hace uso de un tercer número, revelador de las centenas.

El sistema de numeración expuesto se denomina *decimal* o de base diez. Ello se debe a que se utiliza un nuevo dígito para cada *diez* unidades precedentes.

En todo caso, el sistema decimal no es el único sistema de numeración posible. De hecho, muchos pueblos de la antigüedad adoptaron otros sistemas. Sin embargo, bien es cierto que actualmente, el sistema de numeración estándar es el decimal. El ordenador no tiene dedos. Maneja los números bajo el aspecto de impulsos eléctricos; y de forma muy simple: tan sólo detecta la presencia o ausencia de tensión. La manipulación de los datos se efectúa por medio de interruptores: si el interruptor está cerrado pasa electricidad, si está abierto no. Ello equivale a datos del tipo SI o NO (pasa corriente).

A la hora de contar, el ordenador sólo puede adoptar dos estados distintos, lo que significa que puede contar hasta dos directamente. Para cantidades mayores, necesita un nuevo circuito que cuente unidades de dos o "pares". El siguiente circuito contaría pares de pares, esto es unidades de cuatro. Y, en definitiva, aumentando convenientemente el número de circuitos se puede llegar a cualquier cantidad.

El sistema de numeración empleado por los ordenadores se basa, pues, en la detección de dos estados perfectamente diferenciados. Es un sistema en base dos o *binario*. La forma de representar las cantidades en este sistema es análoga a la utilizada para el decimal. El primer dígito de la derecha indica las unidades; cuando se llega a dos se añade un nuevo dígito a la izquierda, etc.

Según esto, para representar a la unidad bastaría con poner el dígito "1". Sin embargo, para dos unidades se ha de utilizar otro dígito adicional. La representación de dos, en binario será "10": el primer dígito (1) indica un par y el segundo (0) cero unidades más.

Análogamente tres se escribe "11", cuatro "100", etc.

El sistema binario sólo exige dos símbolos para elaborar representaciones "0" y "1". Análogamente, el sistema decimal usa diez: del 0 al 9. Otro sistema empleará el número de símbolos indicados por su base. Así, el sistema de numeración de base seis utilizará seis dígitos: del 0 al 5.

A la hora de trabajar con ordenadores resulta útil conocer el sistema binario. Con ello será posible comprender y asimilar mejor los procesos que tienen lugar en su interior.

El sistema binario tiene, no obstante, claros inconvenientes. El primero es la novedad. Estamos acostumbrados al decimal y cualquier otro sistema resulta complicado de manejar, al menos inicialmente.

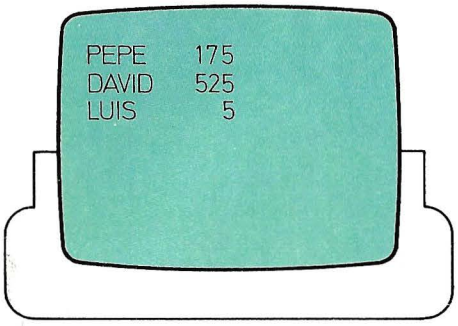
La segunda pega del sistema binario es la extensión de su representación. Para escribir 8 unidades hacen falta cuatro dígitos ("1000"), y para representar 1000 unidades serán necesarios 9 dígitos. Este inconveniente hace tedioso el manejo en binario de grandes cantidades.

La solución al problema consiste en apoyarse en sistemas de numeración alternativos. El sistema ideal sería uno que fuera sencillo de convertir al binario y que, además, su base fuera cercana a la decimal. Para cumplir la primera premisa, es indudable que su base ha de ser una potencia de 2. Las bases que satisfacen ambos requisitos son 4, 8, 16... Las dos más recurridas son la 8 y la 16, que corresponden a los sistemas octal y hexadecimal respectivamente.

El sistema octal emplea ocho símbolos, coincidentes con las cifras decimales del 0 al 7. El paso de octal a binario y viceversa es realmente sencillo. Para ello basta conocer el formato binario de los 8 primeros números en el sistema octal. A su vez, el paso de binario a octal se realiza agrupando los dígitos en conjuntos de tres y traduciéndolos bloque a bloque. Por ejemplo, el número binario 110100111 equivale al octal 647 (110=6, 100=4, 111=7). Para realizar la operación inversa, se ha de "traducir" cada dígito octal por tres binarios, rellenando con tantos ceros a la izquierda como sea necesario. Así, por ejemplo, el número octal 121 corresponde al binario 1010001 (1=1, 2=010, 1=001).

El sistema de numeración hexadecimal necesita 16 símbolos para la representación de sus dígitos. Esto significa que empleando las cifras del 0 al 9 faltan aún seis símbolos para completar las exigencias. El método habitual es utilizar letras para los restantes; las más comúnmente empleadas son las seis primeras mayúsculas: A, B, C, D, E y F. El paso de binario a hexadecimal es similar al que se aplica con el sistema octal, sin más que agrupar esta vez los dígitos binarios de cuatro en cuatro.

Decimal	Binario	Hexadecimal	Octal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	8	10
9	1001	9	11
10	1010	A	12
11	1011	B	13
12	1100	C	14
13	1101	D	15
14	1110	E	16
15	1111	F	17
16	10000	10	20
17	10001	11	21
18	10010	12	22
19	10011	13	23
20	10100	14	24



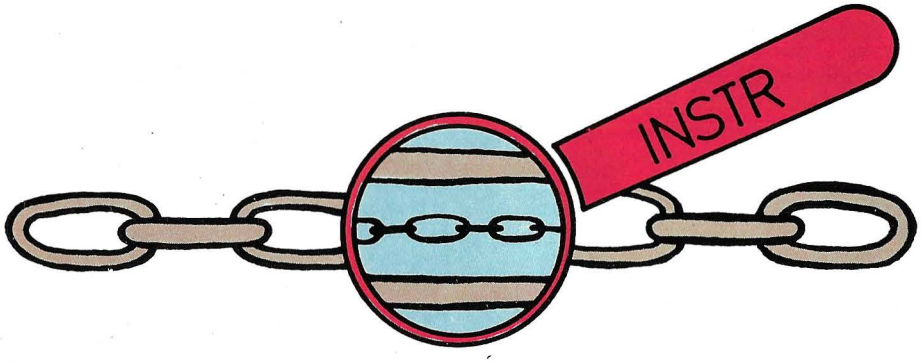
Una de las aplicaciones fundamentales de las funciones STRING$ y SPACE$ es la confección de formatos de pantalla.

ble realizar la misma operación con otros caracteres. Para ello el BASIC dispone de la función STRING$, cuya formulación es la que sigue:

```
STRING$ (<n1>,<n2>) ó
STRING$ (<n1>,<cadena>)
```

Donde <n1> es un número natural que indica la longitud de la cadena y <n2> corresponde al código ASCII del carácter a repetir. En el formato alternativo, <cadena> representa a una cadena de caracteres, cuyo carácter inicial es el que desea repetirse <n1> veces.

Como se observa, esta función admite dos formulaciones alternativas. En ambos casos se crea una cadena formada por la repetición de un carácter. El carácter a repetir coincide, o bien con el que se indica mediante su código ASCII, o con el primero de la cadena que se especifica. El número de veces que se repite el carácter en cuestión viene dado por el valor del parámetro <n1>.



La especialidad de la función INSTR reside en detectar la presencia de una determinada cadena o subcadena dentro de otra.

"/". En la práctica, la mencionada rutina puede simplificarse de la siguiente forma:

```
100 INPUT B$
110 PRINT "/";B$+SPACE$(10-LEN(B$));"/"
```

Ahora, todas las operaciones se realizan en la línea 110.

La posibilidad de formar cadenas de una determinada longitud, integradas por la repetición un solo carácter, no se limita a los espacios en blanco. También es posi-

El mismo resultado del primer formato puede lograrse con la siguiente rutina:

```
 5 INPUT "VECES";I
 8 INPUT "CODIGO ASCII";K
10 LET B$=""
20 FOR J=1 TO I
30 LET B$=B$+CHR$ (K)
40 NEXT J
50 PRINT B$
```

En ella, la variable I contiene el número de caracteres que ha de contener la cadena. Esta variable se utiliza para marcar el límite del bucle FOR. A su vez, K almacena el código ASCII del carácter adecuado. La línea 30, la cual se repite el número de veces estipulado por la variable I, añade el referido carácter a la cadena final cuantas veces sea preciso.

El segundo formato sería algo más complicado de programar; aunque el problema tampoco es excesivo. Veamos una posible solución:

```
 5 INPUT "VECES";I
 8 INPUT "CADENA";X$
10 LET B$=""
20 LET K$=LEFT$ (X$,1)
30 FOR J=1 TO I
40 LET B$=B$+K$
50 NEXT J
60 PRINT B$
```

La diferencia con la primera rutina reside en la línea 20. Aquí es necesario extraer el primer carácter de la cadena introducida en X$.

Un uso elemental, aunque práctico, de la función que nos ocupa, puede ser la simple sustitución de la instrucción que sigue:

```
PRINT "**********************************"
```

por otra bastante más elegante:

```
PRINT STRING$ (32,"*")
```

Con ella se evita la necesidad de escribir 32 veces el asterisco, evitando, además, la posibilidad de cometer algún error a la hora de contar los caracteres a escribir.

INSTR

Función: Busca una secuencia de caracteres dentro de otra, devolviendo como resultado la posición del primer carácter de la secuencia dentro de la cadena.

Formato: INSTR(I,A$,B$)
I: posición a partir de la cual deseamos que se realice la búsqueda.
A$: cadena en la que deseamos buscar la repetición.
B$: secuencia de caracteres que deseamos saber si están contenidos dentro de la cadena A$.

Ejemplos:
```
INSTR(2,X$,P$)          INSTR(C$,F$)
INSTR(5,"CASAS","AS")   INSTR(3,D$,G$)
```

BUSQUEDA DE SUBCONJUNTOS

En lo relativo al manejo de subcadenas o porciones de una cadena de caracteres, se han estudiado en anteriores capítulos

TABLA DE CONVERSION

ORDENADOR	STR$	VAL	SPACE$	STRING$	INSTR	DEFSTR
	STR$ (<número>)	VAL (<cadena>)	SPACE$ (<número>)	STRING$ (<num1>,<num2>)	INSTR ([I,]A$,B$)	DEFSTR
APPLE II (APPLESOFT)	STR$(<número>)	VAL(<cadena>)	—	—	—	—
APRICOT (M-BASIC)	STR$(<número>)	VAL(<cadena>)	SPACE$(<número>)	STRING$(<num1>,<num2>)	INSTR([I,]A$,B$)	DEFSTR
ATARI	STR$(<número>)	VAL(<cadena>)	—	—	—	—
CBM 64	STR$(<número>)	VAL(<cadena>)	—	—	—	—
DRAGON	STR$(<número>)	VAL(<cadena>)	—	STRING$(<num1>,<num2>)	INSTR([I,]A$,B$)	DEFSTR
EQUIPOS MSX	STR$(<número>)	VAL(<cadena>)	SPACE$(<número>)	STRING$(<num1>,<num2>)	INSTR([I,]A$,B$)	DEFSTR
HP-150	STR$(<número>)	VAL(<cadena>)	SPACE$(<número>)	STRING$(<num1>,<num2>)	—	DEFSTR
IBM PC	STR$(<número>)	VAL(<cadena>)	SPACE$(<número>)	STRING$(<num1>,<num2>)	INSTR([I,]A$,B$)	DEFSTR
MPF	STR$(<número>)	VAL(<cadena>)	—	—	—	—
NCR DM-V (MS-BASIC)	STR$(<número>)	VAL(<cadena>)	SPACE$(<número>)	STRING$(<num1>,<num2>)	INSTR([I,]A$,B$)	DEFSTR
NEW BRAIN	STR$(<número>)	VAL(<cadena>)	—	—	INSTR(A$,B$[,I])(1)	—
ORIC	STR$(<número>)	VAL(<cadena>)	—	—	—	—
SHARP MZ-700 (MZ-BASIC)	STR$(<número>)	VAL(<cadena>)	—	—	—	—
SINCLAIR QL	—	—	—	FILL(<cad>, <num>) (2)	A$ INSTR B$ (3)	—
SPECTRAVIDEO	STR$(<número>)	VAL(<cadena>)	SPACE$(<número>)	FILL(<cad>, <num>) (2)	A$ INSTR B$ (3)	DEFSTR
ZX-SPECTRUM	STR$(<número>)	VAL(<cadena>)	—	—	—	—

1. **Función INSTR en el NEW BRAIN**
Busca la cadena B$ en A$ y devuelve la posición del primer carácter de B$ en A$. Si esta cadena no es localizada, devuelve el valor 0.
Si se especifica el número 1, la búsqueda comenzará a partir del carácter situado en la posición 1.
Formato: *INSTR (A$, B$ [, 1)*
A$,B$: cadena de caracteres
I: número entero
Ejemplos: INSTR ("PATATA","TA",2)
INSTR ("CAER","ER")

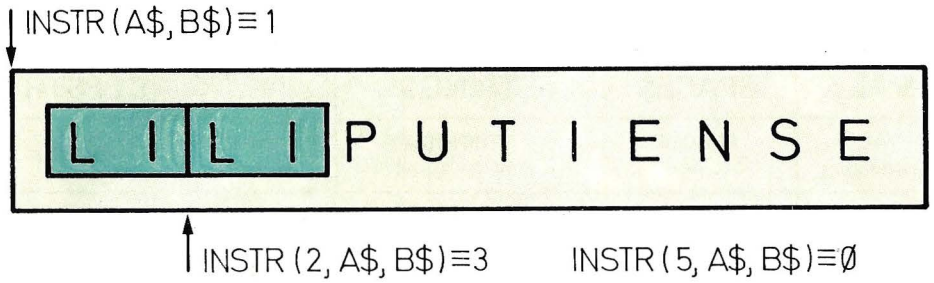
2. **Función FILL$ del QL, equivalente a STRING$**
Crea una cadena de <num> caracteres, formada por la repetición de la cadena de caracteres especificada en <cad>.
Formato: *FILL$ (<cad>, <num>)*
<cad>: cadena de caracteres
<num>: número entero
Ejemplos: PRINT FILL$ ("OA",5) <CR>
OAOAO
PRINT FILL$ ("*",3) <CR>
***

3. **Función INSTR en superBasic QL**
Busca la cadena A$ en B$
Formato: *A$INSTR B$*
Ejemplo: "CA" INSTR "PETACA"

las funciones adecuadas para su extracción: LEFT$, RIGHT$ y MID$. Queda aún otra operación importante que es posible realizar con subcadenas: detectar si una determinada cadena o subcadena está incluida en otra. Esta es una actividad encomendable a la función INSTR, cuyo formato es el siguiente:

INSTR (<n1>,<cadena 1>,<cadena 2>)

En donde <n1> es un número natural, y <cadena 1> y <cadena 2> son dos cade-



Actuación de INSTR bajo diversas formulaciones. La cadena a buscar B$ coincide en el ejemplo con los caracteres "LI".

nas de caracteres. Los siguientes son ejemplos de uso correcto de esta función.

INSTR (A$,H$)
INSTR (5,D$,G$)

Su funcionamiento es el siguiente: la instrucción examinará la < cadena 1> y comprobará si contiene a la <cadena 2>. En el caso de que ello suceda, devolverá como resultado un número. Este corresponderá a la posición que ocupa el primer carácter de la <cadena 2> dentro de la <cadena 1> bajo examen. Si no se encuentra dicha subcadena, el resultado coincidirá con un cero. Idéntica respuesta se obtiene si la cadena en la que se realiza la búsqueda (<cadena 1>) es nula.

```
10 LET A$="COMIENZO"
20 LET B$="MI"
30 PRINT INSTR (A$,B$)
RUN <CR>
3
■
```

En el ejemplo se observa que INSTR devuelve la posición a partir de la cual se encuentra la subcadena "MI" dentro de la cadena inicial "COMIENZO"; en este caso se trata de la posición 3. Este dato puede ser utilizado posteriormente en el argumento de una función MID$ o LEFT$, para trocear la cadena por el punto detectado.
Si se desea iniciar la búsqueda a partir de un carácter distinto del primero, se puede especificar dicho carácter en el argumento de INSTR. Para ello se indicará la posición a partir de la cual debe empezar el proceso de búsqueda: <n 1>.

```
10 LET A$="LILIPUTIENSE"
20 LET B$="LI"
30 PRINT INSTR (A$,B$)
40 PRINT INSTR (3,A$,B$)
RUN <CR>
1
3
■
```

El ejemplo revela la diferencia entre lo que sucede cuando se especifica o no el carácter de partida para el proceso de búsqueda. La instrucción de la línea 30 localiza la subcadena "LI" en el primer carácter; la respuesta que se obtiene coincide, precisamente, con la posición de dicho carácter: 1. Sin embargo, en el segundo caso se prescinde de esta primera subcadena, puesto que no son examinados los dos primeros caracteres de la cadena de origen. Al respecto, hay que tener en cuenta que si el número que se especifica como comienzo de la búsqueda es mayor que la longitud de la cadena, el resultado de la función INSTR será cero.

DEFINICION DE VARIABLES DE CADENA

En un capítulo previo se estudió la forma en la que se podían definir ciertas variables numéricas, especificando su formato. Ello obviaba la necesidad de especificarlo mediante un carácter especial, colocado al final del nombre de la variable (!, % o *). El BASIC ofrece la posibilidad de realizar la misma operación con variables alfanuméricas. En este caso la definición ha de realizarse mediante el comando DEFSTR. Su ejecución evitará la necesidad de incluir el carácter "$" al final de los nombres de variables alfanuméricas previamente definidos. Su formulación presenta el mismo aspecto que la propia de sus homólogos numéricos.

DEFSTR <rango de caracteres>

En la zona <rango de caracteres> se especifican las iniciales de los nombres de variables a definir, separados por comas. Si los caracteres son consecutivos, se puede citar únicamente el primer y último caracteres, separados por un guión. A continuación se incluyen algunos ejemplos relativos al uso de este comando.

DEFSTR A,M–N

Indica que las variables que comiencen por A, M y N serán de cadena de caracteres.

DEFSTR A,C,M–P

En este caso, las variables que comiencen por A, C, M, N, O y P quedan definidas como alfanuméricas.

DEFSTR

Función: Define como cadena de caracteres las variables que comiencen con determinadas letras.

Formato: DEFSTR <rango de caracteres>

Ejemplo:
```
DEFSTR A–D,S,U,X
DEFSTR U
DEFSTR A–F
```

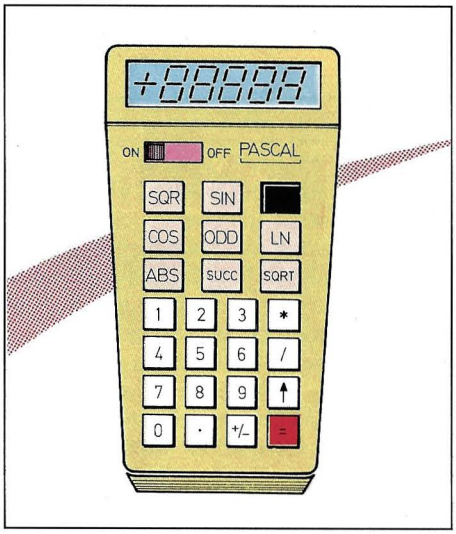
Pascal (7)

Un lenguaje funcional

La modularidad de que gozan los programas codificados en lenguaje PASCAL, no se limita única y exclusivamente a los procedimientos. Hay otros medios que permiten independizar a una determinada zona del programa y englobarla en lo que se denomina un "subprograma". Este subprograma, encargado de realizar una función específica, podrá utilizarse con total comodidad en cualquier otro punto del programa; y sin que por ello haya que escribir de nuevo el mismo conjunto de sentencias cada vez que se desee su intervención.

Para que un bloque de sentencias adquiera la condición de subprograma, no es indispensable que su ejecución deba repetirse en diversos puntos del programa. Aunque su ejecución deba realizarse tan sólo una vez, es recomendable segmentar cuidadosamente el programa en subprogramas que reforzarán su estructura modular. Con ello, el programa resultará más inteligible para cualquier persona que lo lea y, a su vez, será más fácil y rápida su depuración y modificación.

Habitualmente es preciso manipular una determinada información, transformándola según un criterio específico. Suponga, por ejemplo, que se tiene como información de partida los datos BASE y ALTURA (dimensión de la base y altura) de un triángulo. Esta información puede ser manipulada en orden a obtener como resultado el área del triángulo en cuestión. El pequeño subprograma al que se encomiende la función de hallar el área del triángulo, sólo tendrá que tomar los datos de entrada (BASE y ALTURA), obtener su producto, tras ello dividirlo por dos y, por último, asignar este valor resultante a otra variable, por ejemplo: AREA. El referido tratamiento de los datos se podría realizar con la ayuda de un procedimiento con parámetros de entrada y de salida:



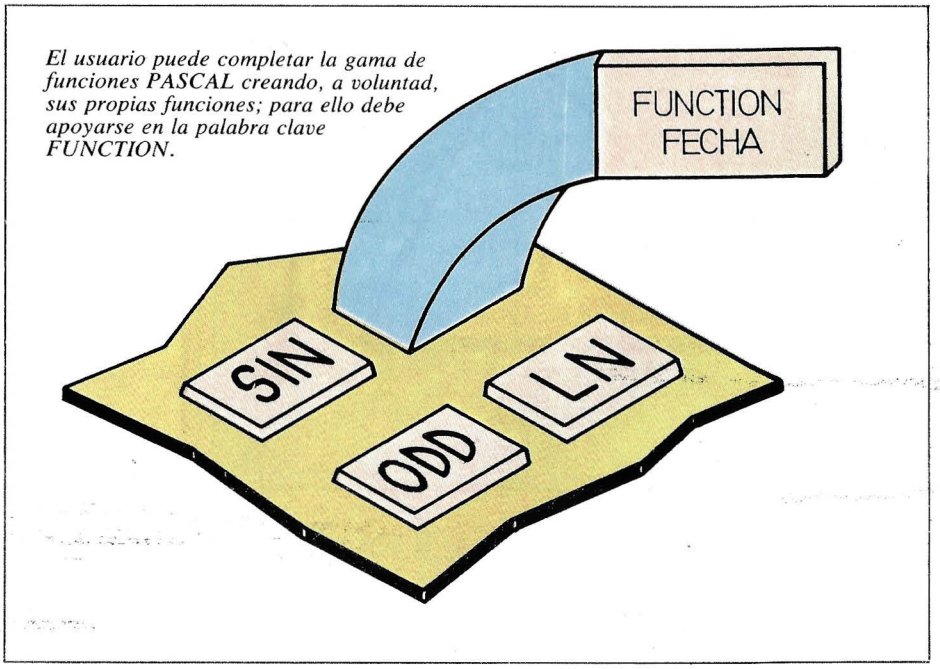
Dentro del repertorio del lenguaje PASCAL, existen toda una serie de funciones ya definidas y a disposición directa del usuario.

```
PROCEDURE areatriángulo (BASE,ALTU-
RA:REAL;VAR AREA:REAL);
BEGIN
  AREA:=BASE*ALTURA/2
END;
```

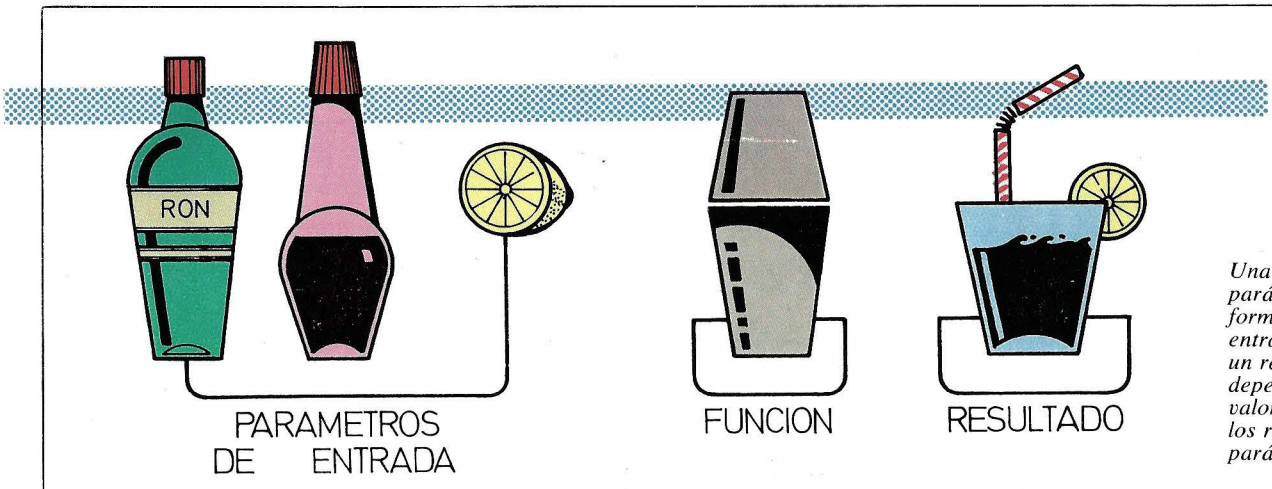
Existe, como ya se apuntó, otra posibilidad para confeccionar este segmento de programa. Esta no es otra que la de definir el referido conjunto de sentencias como una "función", capaz de generar un resultado.

LAS FUNCIONES DEL PASCAL

Una función consiste en una serie de órdenes de cálculo, aplicadas sobre ciertos



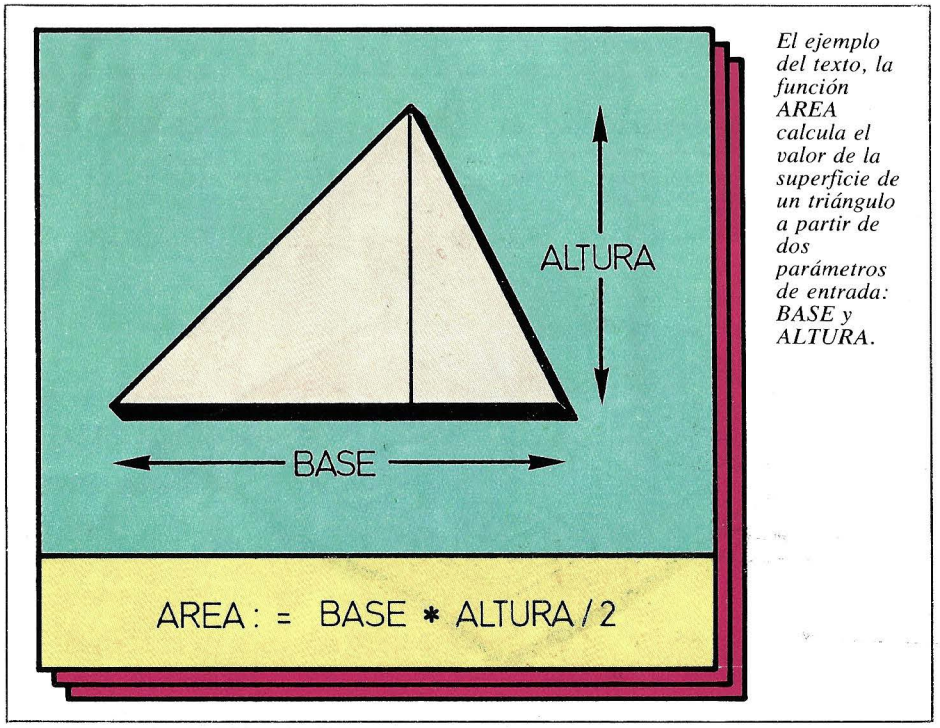
El usuario puede completar la gama de funciones PASCAL creando, a voluntad, sus propias funciones; para ello debe apoyarse en la palabra clave FUNCTION.



Una función con parámetros formales de entrada, obtiene un resultado dependiente del valor actual de los referidos parámetros.

operandos de entrada, que proporcionan un resultado de salida.
Las funciones deben ser declaradas en la correspondiente zona de declaraciones y definiciones del programa; previamente a su uso, al igual que ocurría en el caso de los procedimientos.
La declaración de las funciones se hace de forma similar a la declaración de los procedimientos; si bien, ahora es preciso utilizar la palabra reservada FUNCTION (función) en la cabecera de la declaración. Esta irá seguida por un identificador válido PASCAL, que se utilizará posteriormente para evitar la función, y por una lista opcional de parámetros formales de entrada. Los parámetros aparecerán con sus nombres y sus tipos encerrados entre paréntesis. Por último, tras la lista de parámetros se especificará el tipo de dato del resultado que proporciona la función.
A esta cabecera le seguirá la zona de declaración de variables, en el caso de ser necesario, y el cuerpo de la función. En ella es realmente donde se calcula el resultado a partir de los parámetros actuales de entrada. En definitiva, el aspecto general de una función es el siguiente:

```
FUNCTION <identificador> [(<parámetro
   formal> [,...])]:
<tipo de resultado>;
<zona de declaraciones>;
<cuerpo de la función>;
```



El ejemplo del texto, la función AREA calcula el valor de la superficie de un triángulo a partir de dos parámetros de entrada: BASE y ALTURA.

Comúnmente suele establecerse una división entre las funciones, separándolas en funciones sin parámetros y funciones con parámetros. En realidad, se puede considerar que la lista de parámetros es opcional y, por lo tanto, es facultativo incluirla cuando se considere necesaria u omitir su presencia.
Cabe precisar que una función sólo puede calcular un único valor como resultado, a diferencia de los procedimientos con parámetros que pueden generar más de un resultado de salida.
El resultado debe ser asignado al identificador de la función, dentro del cuerpo de sentencias que la componen. Será esta asignación la que permitirá "devolver" el resultado de la función. Al respecto, resulta obvio que el tipo del valor calculado debe coincidir con el tipo del dato declarado en la cabecera como resultado de la función. Por lo demás, el resultado sólo puede ser de un tipo escalar (REAL, INTEGER, CHAR o BOOLEAN); no es admisible que el resultado de una función sea de tipo estructurado.
El lenguaje PASCAL cuenta con una serie de funciones estandarizadas, cuyo empleo por parte del usuario no exige su declaración previa. Ejemplos de funciones estándar incorporadas al PASCAL y que ya se han mencionado en anteriores capítulos de la obra son: SQR(X), SIN(X), ORD(X), SUCC(X), etc.

Volviendo al ejemplo precedente (cálculo del área de un triángulo), podemos codificarlo ahora por medio de una función:

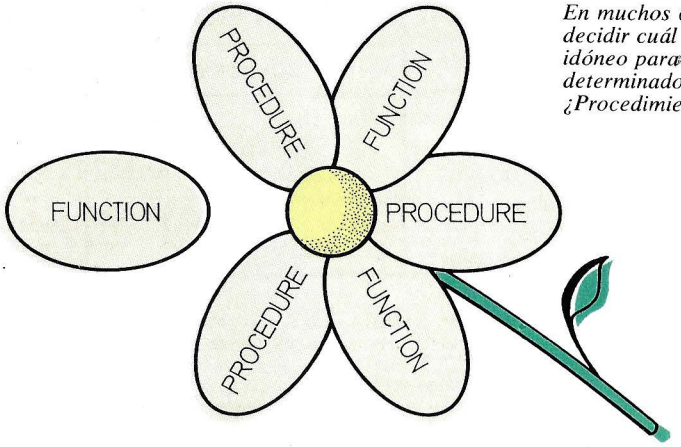
```
PROGRAM AREATRIANGULO (INPUT,
  OUTPUT);
CONST DOS=2;
VAR longbase,longaltura :REAL;
FUNCTION area (BASE,ALTURA: REAL):
  REAL;
BEGIN
area:=BASE*ALTURA/2
END;
BEGIN (*del programa principal*)
  READLN(longbase,longaltura);
  WRITELN('AREA=',area(longbase,
    longaltura)
END.
```

Aquí se observa con total claridad cómo se declara una función para que entregue un resultado de tipo real (REAL).
En el cuerpo de la función, que debe ir escoltado por las palabras BEGIN y END, se asigna el valor calculado al identificador "area". Tal asignación permitirá que la función "devuelva" el valor calculado al punto de llamada.
La llamada o activación de una función PASCAL, se puede realizar de dos formas. La primera se concreta en una llamada directa, sin más que nombrar en el momento adecuado el identificador de la función, seguido por la lista de parámetros actuales si los hubiere. Así es como tiene lugar en el ejemplo anterior: la llamada se realiza dentro de una sentencia de escritura que presentará directamente el resultado de la función. En el segundo caso, la llamada se realiza dentro de una sentencia de asignación o de comparación; por ejemplo:

```
AREADELTRIANGULO:=area(BASE,
  ALTURA);
```

"FUNCTION" VERSUS "PROCEDURE"

Cabe pensar que cualquier segmento de programa codificable a modo de procedimiento, puede codificarse análogamente mediante una función. Tal afirmación es cierta, debido a la gran similitud que existe



En muchos casos es difícil decidir cuál es el método idóneo para resolver un determinado problema... ¿Procedimiento o función?..

Ambito de las variables

Para que una variable PASCAL pueda ser utilizada en el contexto de un programa, debe ser declarada con anterioridad su intervención. Esta declaración puede realizarse en distintas zonas de un programa: en la zona de declaraciones del propio programa, en la zona de declaraciones de una función o en la de un procedimiento. Ello hará que el ámbito de utilización de esa variable se vea restringido al bloque en el que ha sido declarada.
Por ejemplo, si una variable se declara a nivel de programa, es decir en la zona que sigue al bloque PROGRAM, se podrá referenciar dentro de cualquier subprograma incluido en él, y también fuera de ellos. Este tipo de variables se denominan *globales*.
Por el contrario, si una variable se declara dentro de una determinada función o procedimiento, ésta será de tipo *local:* sus valores sólo estarán disponibles dentro de dicho procedimiento o función y nunca en una zona del programa exterior al mismo. Así, por ejemplo, una variable puede ser global sólo en dos o más subprogramas, local en alguno de ellos o global a todos los subprogramas.
En el ejemplo de la figura, las variables J y K son globales a todo el programa cuyo nombre es "externo"; sus valores también estarán disponibles en el interior de la función denominada "interna". A su vez, las variables L y M son locales en la función "interna" y no serán reconocidas fuera de ella. Por el contrario, la variable I tendrá valores distintos fuera y dentro de la función "interna", ya que está declarada en ella como variable local y como variable global en el programa "externo".
La declaración de una variable local con el mismo identificador que una global no producirá error alguno, si bien los valores asignados a ella en cada zona del programa serán distintos. En efecto, aunque compartan el mismo nombre, representan a variables distintas.
Esta característica suele ser fuente de numerosos errores por parte de programadores poco experimentados: es muy fácil intentar utilizar una variable local fuera de su ámbito, o bien querer disponer del valor de una variable global en un bloque en el que está definida otra variable local con el mismo identificador.
Para conocer el ámbito de una variable hay que examinar la zona de declaraciones del bloque en el que esté inmersa; si no está declarada en él, habrá que examinar el bloque que contenga al anterior, y así sucesivamente hasta llegar al bloque más general que coincidirá con el PROGRAM. Si tampoco está definida en él, resultará obvio que dicha variable generará un error de complicación al no estar declarada en ninguna zona del programa.

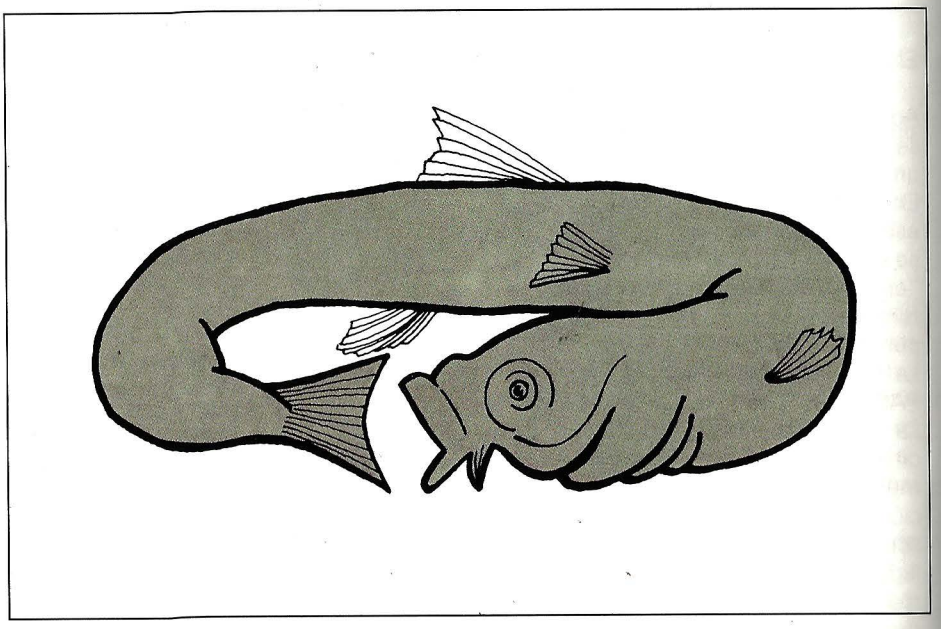
```
PROGRAM externo (INPUT,OUTPUT);
VAR I,J,K :REAL;
—
—
  FUNCTION interna (...);
  VAR I,L,M :REAL;
  —
  —
  END;
BEGIN
—
—
END.
```

entre ambos conceptos. De ahí que quepa la duda de cuándo utilizar un procedimiento (PROCEDURE) y cuándo una función (FUNCTION).
En términos prácticos existen claras diferencias entre procedimiento y función. De entrada, una función sólo puede generar un único resultado, mientras que un procedimiento puede entregar cualquier número de valores de salida (todos, uno, varios o ninguno). Por otro lado, los parámetros formales de una función sólo son de entrada —aportan valores a la función—, mientras que los parámetros de un procedimiento pueden ser de entrada y salida de datos.
Para precisar el uso de uno u otro método en cada caso específico, es conveniente recordar la siguiente regla. Cuando sólo sea necesario un único resultado, teniendo o no parámetros de entrada, se debe utilizar una FUNCTION. En los demás casos, esto es, cuando no haya resultado o haya más de uno, se utilizará el PROCEDURE. Por supuesto que esta regla no es inquebrantable, pero su aplicación facilitará, sin lugar a dudas, un uso más eficaz de los recursos que brinda el lenguaje PASCAL.

TABLA DE COMANDOS-PASCAL

COMANDO	COMETIDO	OBSERVACIONES
FUNCTION <id.1>{(<lista>)}: <tipo>;	Declaración como función de un conjunto de sentencias	Zona de declaraciones y definiciones
<id.1>,...:<tipo>{,<id.2>,...: <tipo>,...}	Declaración de los parámetros formales de una función	Cabecera de una FUNCTION

NOTA: <id.>, <id.1>, <id.2>: identificadores válidos PASCAL. <lista>: lista de parámetros formales de entrada de una función. <tipo>: tipo de dato del resultado, que no podrá ser estructurado.



Las estructuras recursivas reflejan en el ámbito informático la imagen de la pescadilla que se muerde la cola: ¿se están alimentando o sirven de alimento?

LA RECURSIVIDAD DEL PASCAL

La recursividad consiste en la facultad de un subprograma para llamarse a sí mismo. En efecto, dentro de un procedimiento o función se puede incluir una llamada a ese mismo procedimiento o función. Para clarificar este concepto, puede examinarse la función siguiente, que calculará de forma recursiva el factorial de un número entero. (El factorial de un número se define como el producto de todos los enteros comprendidos entre 1 y dicho número, ambos inclusive).

```
FUNCTION factorial (N:INTERGER):
  INTERGER;
BEGIN
  IF N=1 THEN factorial:=1
    ELSE factorial:=factorial(N-1)*N
END;
```

En el ejemplo se observa cómo dentro del cuerpo de la función "factorial" se efectúa una nueva llamada a la misma función "factorial", aunque ahora con otro parámetro actual de entrada. Suponga que en un determinado programa se realiza la siguiente llamada a esta función:

```
NUMERO:=factorial(3);
```

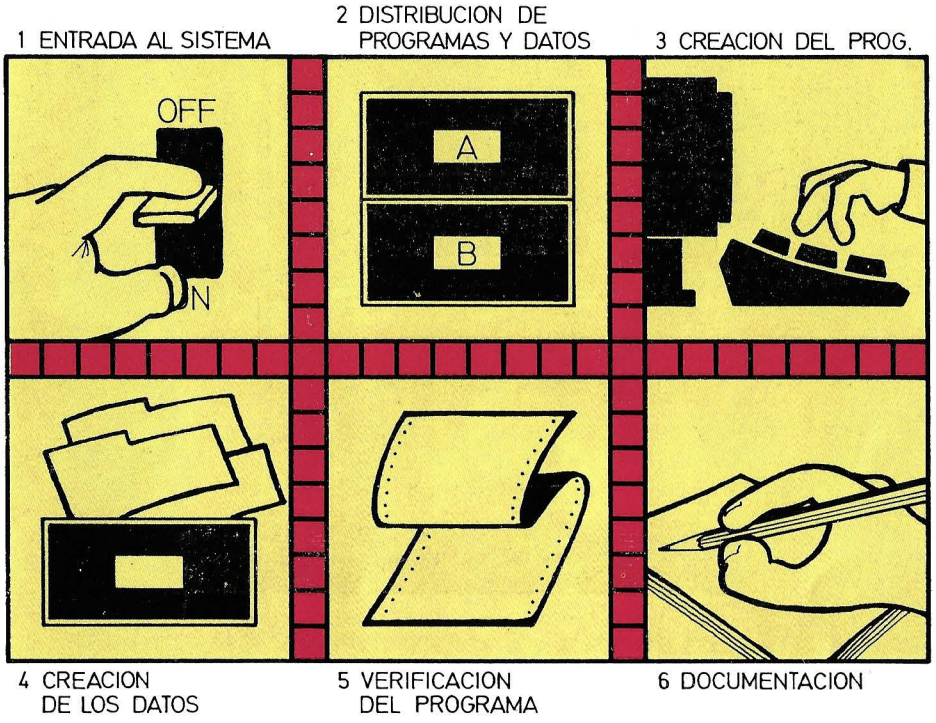
Al intentar obtener el resultado para el valor actual (3) del parámetro formal "N", la función detectará que dicho valor es distinto de 1 (IF N=1 . . .). A raíz de ello, se producirá una nueva llamada a la función, ahora con el valor de entrada 2, dejando pendiente el producto de 3 por el resultado de la función "factorial (2)". La nueva llamada se encontrará con una situación parecida a la del caso anterior, operándose una nueva llamada a la función factorial, esta vez con el parámetro actual 1. De nuevo, quedará pendiente el producto del 2 por el resultado de la función "factorial (1)". La llamada "factorial (1)" cerrará este encadenamiento de llamadas, devolviendo ahora el valor 1 al ser cierta la condición impuesta en la sentencia IF, con lo que se podrán realizar todas las operaciones pendientes de las llamadas anteriores.
Dentro de una estructura recursiva debe figurar siempre alguna sentencia de ruptura; de lo contrario se formaría un ciclo cerrado sin final. En él se repetiría incesantemente la llamada a la función o al procedimiento, hasta que se originara un error por desbordamiento (overflow) de alguna de las variables implicadas en la estructura recursiva.

MS/DOS (9)

Desarrollo de programas, bajo el control del MS/DOS

El desarrollo de programas por parte del usuario es una faceta a la que quizá no se preste la atención que merece, debido a la existencia en el mercado de una gran cantidad de programas que cubren casi todas las necesidades (hojas electrónicas, tratamientos de texto, paquetes gráficos, gestión de archivos y bases de datos, contabilidad, etc.). De ahí que la tarea de desarrollar programas se reduzca, por lo general, a unas cuantas aplicaciones sencillas y muy específicas, para el uso particular de quien acomete su diseño.
En cualquier caso, estas aplicaciones suelen elaborarse empleando métodos poco disciplinados y ortodoxos; en parte debido a que suelen estar pensados para uso muy personal.

DESARROLLO DE PROGRAMAS

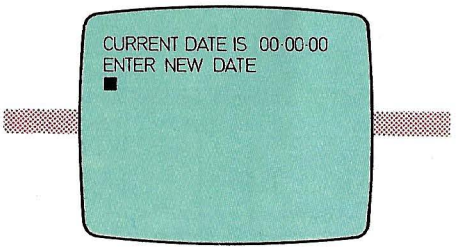


Secuencia de pasos a poner en práctica para el correcto y completo desarrollo de un programa.

Aunque el volumen de programas de creación propia no acostumbre a alcanzar un nivel considerable, siempre es buena técnica disponer de una metodología con la cual llevarlos a cabo. Ello frenará el impulso de querer ultimar los programas inmediatamente después de sentarse tras el teclado.
A continuación se enumeran una serie de pasos a contemplar para el completo y correcto desarrollo de un programa: desde la entrada al ordenador, hasta la conclusión eficaz del programa y su posterior documentación. La descripción versará en torno a un ejemplo concebido para el ordenador personal IBM-PC XT, dotado del sistema operativo MS/DOS (PC/DOS) versión 2.00.

- *La entrada al sistema*

Como ya es sabido, al conectar el ordenador entra inmediatamente en funcionamiento un programa cuya misión es copiar el sistema operativo, desde el disco en el que reside, hasta la memoria del ordenador. Al finalizar este proceso se puede trabajar ya bajo el control del sistema operativo. Cabe recordar que es altamente recomendable disponer siempre de una copia de respaldo del sistema operativo para evitar disgustos en el caso de que el disco que lo contiene sufra daños irreparables. Acto seguido, el MS/DOS pedirá la fecha y hora con los formatos siguientes:



La introducción de la fecha y hora actuales en el ordenador, es una medida muy recomendable. Ello permitirá al usuario contar con una información cronológica exacta, relativa a la creación o última modificación de los archivos manipulados.

Current date is (DD-MM-YY) : 00-00-0000
Enter new date
22-03-1985
Current time 0:0:0
Enter new time
10:00:00

(Los datos que figuran en cursiva corresponden a las respuestas que proporciona el usuario).

De no especificar la fecha y la hora, el sistema adoptará los datos horarios correspondientes al instante en el que terminó la sesión anterior. En tal caso se perderán las ventajas de disponer como información adicional de la fecha y hora exactas de la creación o última modificación de los ficheros.

Antes de proseguir con el estudio, conviene recordar que los textos que aparezcan en letra cursiva, dentro de los ejemplos, serán introducidos por el usuario, e irán seguidos por una acción sobre la tecla "ENTER" o "RETURN".

● *Distribución de programas y datos*

En el caso de que se disponga de una revisión del MS/DOS coincidente o posterior a la 2.00, se puede aprovechar el uso de directorios y subdirectorios para estructurar mejor el espacio en disco, y almacenar agrupadamente los diversos tipos de ficheros. En este orden, es interesante poder definir varios directorios en los que se agruparán sólo programas, sólo datos, o sólo utilidades, etc...

En nuestro caso se creará un directorio para los programas (PROG) y otro para albergar los archivos de datos y resultados (DATA).

Así pues, mediante el comando MKDIR o MD se crearán los dos directorios antes citados. Previamente, seleccionaremos el disco introducido en la unidad A como disco de trabajo; el disco alojado en la unidad C quedará reservado a los programas de uso general. Las acciones a realizar son las ordenadas por el siguiente grupo de comandos:

C> *A:* (cambia la unidad de disco tomada por defecto)
A> *MD PROG* (crea el directorio PROG)
A> *MD DATA* (crea el directorio DATA)

El contenido del disquete puede examinarse apelando al comando DIR:

A> *DIR A:*
PROG <DIR> 23-03-85 10:01:27
DATA <DIR> 23-03-85 10:01:30

● *Creación del programa*

Una vez creados los directorios, habrá que dirigirse al directorio PROG para crear en su ámbito el programa BASIC. El comando que permite cambiar de directorio es CHDIR o CD:

A> *CD PROG*

El contenido del directorio PROG es:

A> *DIR*
. <DIR> 23-03-85 10:01:27
. .<DIR> 23-03-85 10:01:27

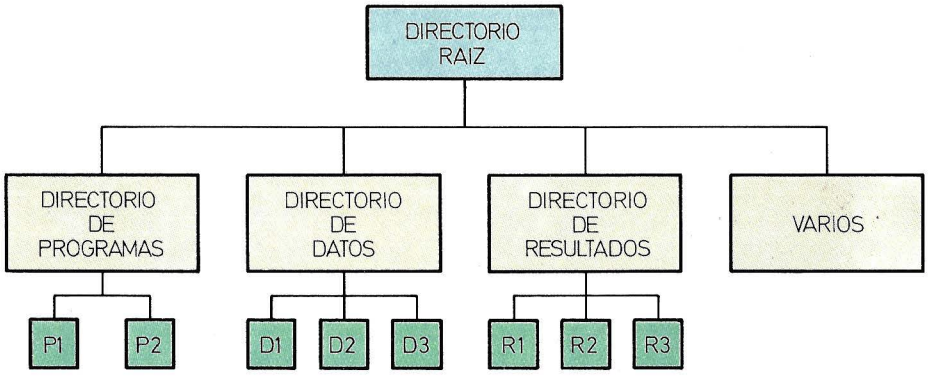
Los signos "." y ".." son dos entradas que se utilizan para referenciar al directorio actual (.) y al directorio raíz (..).

El programa BASIC se confeccionará con la colaboración del editor asociado al traductor BASIC; éste ofrece mayores ventajas al efecto que el editor de líneas EDLIN que acompaña al MS/DOS. Para invocar al BASIC bastará con ejecutar el siguiente comando:

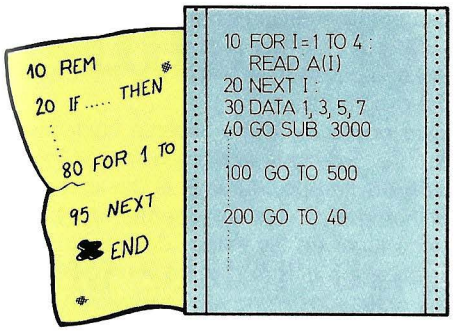
A> *C:BASICA*

El programa cuyas sucesivas líneas habrá que introducir, puede coincidir, por ejemplo, con el que aparece en el cuadro adjunto. Su cometido se reduce a leer números de un fichero, ordenarlos y, posteriormente, escribirlos en otro fichero; ambos ficheros afectados se encuentran en el directorio DATA. Al terminar la introducción del programa a través del teclado,

La adopción de una metodología clara y sencilla a la hora de proceder al desarrollo de programas, reduce las probabilidades de error, facilita el trabajo y contribuye a aprovechar con mayor eficacia los recursos que brinda el ordenador.



La compartimentación en directorios de los diversos tipos de ficheros, facilita las tareas de mantenimiento y localización de las aplicaciones.



A la hora de crear el programa, es conveniente redactarlo de una forma clara y estructurada. Ello facilitará su lectura y minimizará el tiempo necesario para su corrección y puesta a punto.

conviene almacenarlo en disco utilizando la orden:

OK *SAVE "A: ORDENA.BAS", A*

El parámetro ", A" especifica que la grabación se realice en el formato ASCII. Concluida la grabación, se regresará al control del sistema operativo apelando al comando al efecto:

OK *SYSTEM*

- *Creación de los datos*

Como se ha convenido anteriormente, los datos de entrada al programa estarán localizados en un fichero incluido en el directorio DATA. En consecuencia, es necesario dirigirse al directorio en cuestión:

A> *CD\DATA*

Por supuesto, también es posible editar directamente desde el emplazamiento actual, aunque tomando la precaución de indicar el camino que conduce al directorio DATA:

A> *C:EDLIN\DATA\LIO.DAT*

Por razones de simple comodidad, optaremos en nuestro ejemplo por este último camino; así se evitará la necesidad de saltar continuamente de un directorio a otro. Cabe observar que ahora se emplea el editor de líneas EDLIN, ya que en este caso no nos sirve el editor del BASIC.
Acto seguido, se introducirán los datos necesarios: cantidad de datos a ordenar y números que deben someterse al proceso de ordenación, uno por línea de fichero. Una vez introducidos todos ellos se grabará el fichero en cuestión, tal y como ilustra la figura adjunta.

- *Verificación del programa*

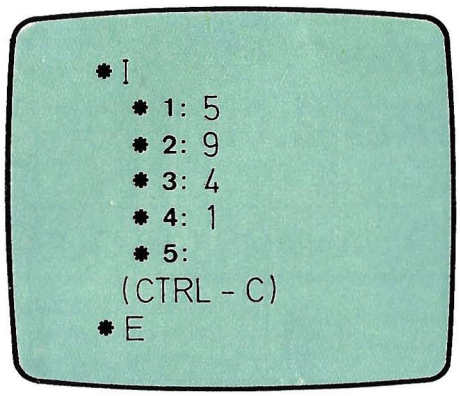
En este momento se han creado ya los

Mejoras y novedades del MS/DOS versión 2.00

La aparición de las versiones 2.00 y sucesivas del sistema operativo MS/DOS han hecho que éste adquiera progresivamente una mayor potencia y versatilidad.
Un hecho frecuente es que la versión del sistema operativo que equipa al ordenador en el momento de adquirirlo, quede superada al cabo de un cierto tiempo por nuevas revisiones. En muchos casos, la adopción de una nueva versión del sistema operativo se retrasa excesivamente por lo tedioso que puede resultar la tarea de actualizar la configuración.
En este punto cabe señalar que el MS/DOS permite un cambio de configuración sencillo cada vez que se arranca el sistema. Ello se debe a que es un fichero editable el que incluye a los parámetros de configuración, tales como los nombres de los dispositivos, las características de los discos y otra información adicional relativa al modo de operación del MS/DOS.
Una diferencia sustancial que caracteriza al MS/DOS versión 2.00, queda reflejada en la posibilidad de trabajar con discos rígidos. Estos permiten un acceso mucho más rápido a la información almacenada, al tiempo que incrementan espectacularmente la capacidad de almacenamiento externo. En este mismo entorno, hay que citar la posibilidad de trabajar con directorios jerárquicos, organizando los ficheros en base a estructuras arborescentes. Con ello se logrará organizar la información de una forma más eficiente.
En cuanto a la flexibilidad de operación, cabe destacar el redireccionamiento de las salidas y entradas por pantalla y teclado a otros dispositivos asociados al sistema; la posibilidad de que un programa adopte automáticamente los datos de salida de otro programa como datos de entrada propios; el más eficaz control de la pantalla por medio de secuencias de caracteres especiales que controlan la posición del cursor; el trabajo con color; las funciones de las teclas definibles; etc.
Por último, hay que hacer mención a los nuevos comandos que aporta la versión 2.00 y a los ya existentes que se ven notablemente modificados en la nueva versión; todos ellos quedan reflejados en el cuadro adjunto.

COMANDOS MS/DOS

Nuevos comandos del MS/DOS 2.00	Comandos modificados respecto a anteriores versiones
ASSIGN	CHKDSK
BACKUP	COMP
BREAK	DEBUG
CLS	DIR
CTTY	DISKCOPY
ECHO	DISKCOMP
IF	EDLIN
FOR	ERASE
SHIFT	FORMAT
GOTO	LPT2
FDISK	LPT3
GRAPHICS	COM2
MKDIR	
RMDIR	
CHDIR	
PATH	
PRINT	
PROMPT	
RECOVER	
RESTORE	
SET	
TREE	
VER	
VERIFY	
VOL	



La introducción de los datos con el apoyo del editor EDLIN, empieza con la orden de inserción. Tras introducir los datos oportunos, se accionarán simultáneamente las teclas CONTROL y C para dar por concluida la edición. Los datos serán grabados al concluir la sesión con la orden E.

para realizar la carga del programa y ejecutarlo a continuación; si bien, antes de ejecutarlo, se obtendrá un listado del programa para examinar su aspecto actual:

```
A> C:BASICA
OK LOAD "ORDENA.BAS"
OK LIST
    10 REM PROGRAMA PARA
    ORDENAR NUMEROS
     .
     .
     .
    300 END
OK RUN
```

Llegados a este punto, pueden haber ocurrido cuatro cosas:
a) Error en la lectura de datos.
b) El programa no funciona correctamente.

```
10  REM PROGRAMA PARA ORDENAR NUMEROS
20  REM APERTURA DEL FICHERO DE DATOS Y DE RESULTADOS
30  OPEN "\DATA\LIO.DAT" FOR INPUT AS #1
40  OPEN "\DATA\ORDEN.DAT" FOR OUTPUT AS #2
50  REM LECTURA DE DATOS
60  INPUT #1, NPUN
70  DIM NUM(NPUN)
80  FOR I=1 TO NPUN
90      INPUT #1, NUM(I)
100 NEXT I
110 REM ORDENACION DE LOS DATOS
120 FOR I=1 TO NPUN-1
130     FOR J=I+1 TO NPUN
140         IF NUM(J) > NUM(I) THEN GOTO 180
150         PASO = NUM(J)
160         NUM(J) = NUM(I)
170         NUM(I) = PASO
180     NEXT J
190 NEXT I
200 REM ESCRITURA EN EL ARCHIVO DE RESULTADOS
210 WRITE #2, NPUN
220 FOR I=1 TO NPUN
230     WRITE #2, NUM(I)
240 NEXT I
250 REM CIERRE DE LOS FICHEROS
260 CLOSE #1
270 CLOSE #2
280 STOP
300 END
```

elementos necesarios para que sea posible llevar a cabo la ordenación. El siguiente paso debe coincidir, pues, con la ejecución del programa.
Habrá que invocar de nuevo al BASIC,

c) Error en la escritura de resultados.
d) Todo funciona correctamente.
En cualquiera de los tres primeros casos habrá que revisar el programa y/o los datos, regresando a los pasos de creación de datos y creación del programa. Una vez corregidos los posibles errores, se repetirá el proceso hasta llegar a la verificación del programa.
Esta operación debe hacerse cuidadosamente. Sin duda alguna, la mayor inversión en tiempo, durante el desarrollo de un programa medianamente complejo, tiene lugar en las etapas de corrección y verificación del mismo. La verificación de los resultados exige su visualización en la pantalla. Para ello será preciso volver al sistema operativo:

```
OK SYSTEM
A> C:PRINT\DATA\ORDEN.DAT
```

Aunque el programa se haya ejecutado correctamente, ello no garantiza que esté totalmente exento de errores. Es aconsejable ejecutarlo varias veces, evaluando casos extremos para comprobar la plena corrección de su funcionamiento.

- *Documentación*

El último paso, aunque no por ello menos importante, consiste en documentar el programa. Esta documentación debe contener los elementos mínimos que garanticen al usuario una correcta operación con el programa (descripción de los datos de entrada y salida, consideraciones especiales, ubicación, etc.). Debe incluir además una breve descripción de su funcionamiento interno, algoritmos que utiliza y límites de aplicación.
De esta forma, el programa estará siempre dispuesto para su posible explotación y podrá ser modificado posteriormente por otra persona distinta de quien lo confeccionó, sin que ello suponga un esfuerzo excesivo.

CONSIDERACIONES FINALES

El supuesto descrito a través de los párrafos precedentes, intenta dar una visión general de los pasos a seguir para el desarrollo armónico y estructurado de programas. Desde luego, no se trata de una guía que deba seguir a pies juntillas, puesto que en otros casos que no compartan la estructura del indicado como ejemplo, su aplicación puede ser incluso contraproducente. No obstante, su filosofía general puede aplicarse a la mayor parte de las situaciones, ayudando a un mejor mantenimiento y uso de la información.

Multitexto (y 3)

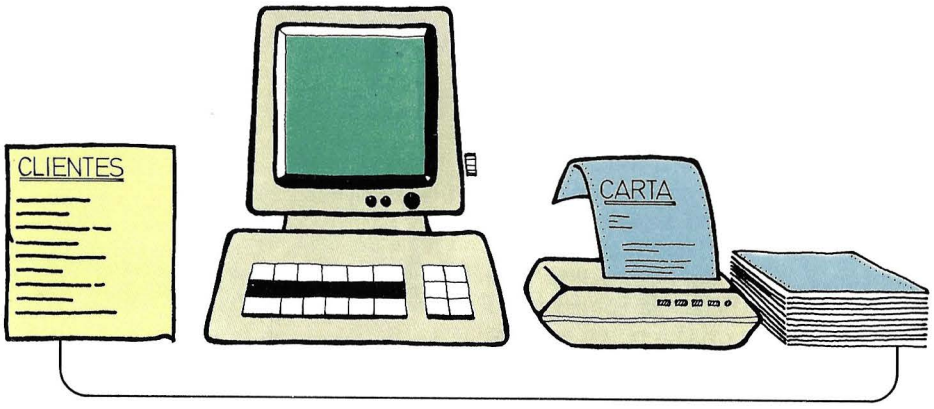
Una sesión de trabajo

Para evaluar las posibilidades prácticas del programa MULTITEXTO, se reproducirá en el presente capítulo una sesión de trabajo con este procesador de textos. En esta ocasión, nuestro esfuerzo se concretará en la producción de un documento síntesis entre dos archivos independientes.

OBJETIVOS DE LA SESION

El supuesto de trabajo parte de una empresa que desea convocar a una reunión a todos sus clientes. Para ello se utilizará un ordenador personal con el que, mediante el programa MULTITEXTO, se debe producir una carta personalizada para cada uno de ellos. Una primera solución consiste sencillamente en producir un documento distinto para cada cliente, aunque ello supondrá para el usuario una labor tediosa y lenta. Otra solución, más aceptable, consiste en redactar una carta-tipo, en la que se incluirán el texto básico de la carta, dejando huecos en blanco en las posiciones reservadas para los datos personales de cada cliente. A continuación, se pueden obtener copias de este documento inicial, incluyendo en los huecos en blanco los datos de cada cliente. Con esta solución, el trabajo del usuario se ve simplificado al no verse obligado a repetir en cada carta el texto básico. No obstante, el proceso sigue siendo lento y demasiado manual. Por lo demás, si después de haber producido todas las cartas se desea modificar alguno de los datos fijos, como puede ser la hora, fecha o lugar de la reunión, será preciso repetir todo el proceso.

El procesador de textos MULTITEXTO dispone de una función denominada FUSION, que sirve precisamente para lo que

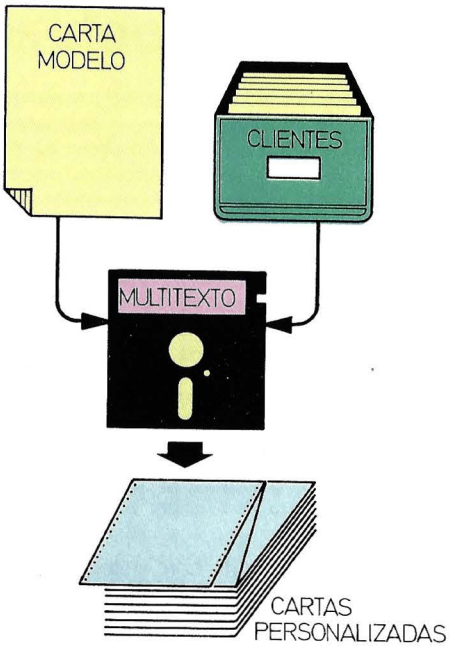


La producción de cartas personalizadas puede solventarse aplicando diversos métodos, más o menos rápidos y, en definitiva, eficaces. Sin lugar a dudas, la solución idónea la aporta el ordenador equipado con un programa capaz de personalizar las cartas de forma automática.

su nombre indica: fusionar en un único escrito dos documentos distintos. Haciendo uso de esta función se puede encontrar una solución idónea al problema planteado. Por lo tanto se almacenará en un documento primario el texto-tipo de la carta y, por otro lado, en un documento secundario, archivaremos los datos personales de los clientes a convocar. Tras ello, y utilizando la función FUSION del MULTITEXTO, será posible confeccionar las cartas personalizadas de forma automática. Si más adelante se desea modificar el texto base o los datos de los clientes, la repetición de las cartas se realizará inmediatamente, sin más que modificar el documento apropiado.

PREPARACION DE LOS DOCUMENTOS BASICOS

Después de conectar el ordenador personal e invocar al programa MULTITEXTO, aparecerá en la pantalla el menú principal



El paquete MULTITEXTO es capaz de producir documentos finales resultantes de combinar dos archivos independientes. Ello constituye el medio idóneo para la edición de cartas personalizadas.

```
                    CREAR UN DOCUMENTO NUEVO

              Introduzca el Nombre del Documento Nuevo a Crear

Drive: A          Documento: CONVOCATORIA

Aproximadamente 00077824 caracteres [00031 Página(s)] disponibles AAA


          Pulse (return) para seguir, (PgDn) para cambiar de drive
(Ctrl) (Home) Directorio por Defecto / (Ctrl) (End) Ir al Siguiente Directorio

                                                        S: ↓  N: ↓
```

```
                    CREAR UN DOCUMENTO NUEVO

              Introduzca el Nombre del Documento Nuevo a Crear

Drive: A          Documento: CLIENTES

Aproximadamente 00077824 caracteres [00031 Página(s)] disponibles AAA


          Pulse (return) para seguir, (PgDn) para cambiar de drive
(Ctrl) (Home) Directorio por Defecto / (Ctrl) (End) Ir al Siguiente Directorio

                                                        S: ↓  N: ↓
```

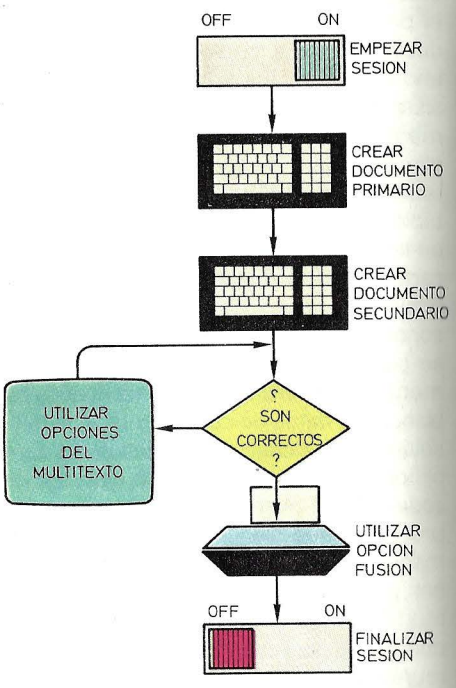
```
                      FUSION DE DOCUMENTOS

DOCUMENTO PRIMARIO                DOCUMENTO SECUNDARIO
Drive:    A                       Drive:    A
Nombre: CONVOCATORIA              Nombre: CLIENTES

Aproximadamente 00075776 caracteres [00030 Página(s)] disponibles AAA        BBB

          Seguir (F10), Abortar (Esc). Cambiar de Drive (PgDn)
(Ctrl) (Home) Directorio por Defecto / (Ctrl) (End) Ir al Siguiente Directorio

                                                        S: ↓  N: ↓
```

La sesión práctica que se describe en el presente artículo supone el uso de las opciones 2 y 5 del menú principal del MULTITEXTO. En primer lugar, el usuario debe crear el documento que denominamos CONVOCATORIA; éste incluirá el esqueleto de la carta que debe remitirse a los clientes a convocar.
A continuación, es preciso crear el documento CLIENTES, con los datos personales de los clientes a convocar.
Una vez creados los documentos primario y secundario, habrá que utilizar la opción 5 del menú principal (imprimir un documento fusión). En la pantalla asociada a dicha opción, habrá que especificar los nombres de los documentos a fusionar en su estricto orden (primario y secundario).

de opciones. Si previamente no disponíamos de documentos con el texto de la carta ni con los datos personales de los clientes, la primera selección a realizar debe ser la opción 2: "CREAR UN DOCUMENTO NUEVO". Tras pulsar la tecla RETURN aparecerá una nueva pantalla en la que especificaremos el nombre del documento primario; en nuestro caso, éste puede ser CONVOCATORIA para el documento primario que almacenará el "esqueleto" de la carta, y CLIENTES para el documento secundario en el que introduciremos los datos personales de los convocados a la reunión. A continuación, la pantalla aparecerá en blanco y el programa aguardará a la introducción de los correspondientes textos. La única novedad a tener en cuenta respecto a la introducción clásica de caracteres mediante un programa para el tratamiento de textos, reside en la sintaxis que indicará los nombres de los puntos de fusión del documento secundario.

Supongamos que los datos personales necesarios para cada cliente son el NOMBRE, los dos APELLIDOS y su DIRECCION, y que el formato de la carta debe comenzar con un encabezamiento en el que se especifique el NOMBRE, los APELLIDOS y la DIRECCION del cliente. Acto seguido, se desea incluir la frase: "Querido NOMBRE:" y después "TENGO EL GUSTO DE INVITARLE A LA REUNION QUE CELEBRAREMOS...". En tal caso, será preciso introducir en el documento primario lo siguiente:

- En la primera línea: ⊢ NOM ⊢ ⊢AP1 ⊢ ⊢ AP2 ⊢ con lo que se hará referencia a los datos NOM, AP1 y AP2 que serán extraídos del fichero secundario al imprimir el documento de fusión.
- En la segunda línea: ⊢ DIR ⊢. Esta es una referencia a la variable DIR que extraerá la dirección del cliente del documento secundario.
- En la tercera línea: Querido ⊢ NOM ⊢:. De nuevo, utilizamos la variable NOM para repetir el nombre del cliente en la



Secuencia de pasos a poner en práctica para la edición de un documento que sintetice el contenido de dos archivos. En el caso que nos ocupa, el resultado es la edición de un conjunto de cartas personalizadas.

primera línea de la carta; además, en este caso, estamos mezclando una palabra ("Querido") con una variable (NOM).

- En las restantes líneas: "Tengo el gusto de invitarle a la reunión que celebraremos...", hasta terminar el esqueleto de la carta.

Obsérvese que cada vez que en el documento primario hacemos referencia a un dato "traducible", lo encerramos entre dos caracteres "⊢"; de esta forma, el programa es capaz de diferenciar el texto básico del documento primario del texto que debe ser sustituido por información del documento secundario. Una vez finalizada la creación del documento primario, el usuario debe elaborar el documento de CLIENTES utilizando, de nuevo, la opción 2 del menú principal. Para ello introducirá:

- En la primera línea: ⊢ NOM ⊢<< Julián ⊢ << ⊢ AP1 ⊢ << Rodríguez ⊢ << ⊢ AP2 ⊢ << García ⊢ << ⊢ DIR ⊢ << C/Ríos Rosas, 27. 28015-MADRID ⊢ <<.

De esta forma se asigna un dato específico a cada una de las variables referenciadas en el documento primario.

- En las restantes líneas se introducirán con el mismo formato, los datos relativos a los restantes clientes.

El documento CLIENTES podrá ser utilizado para obtener una fusión con el documento CONVOCATORIA, o con cualquier otro que utilice los mismos nombres como puntos de fusión. De esta foma, se dispone de un archivo maestro de clientes, que será utilizado para producir cualquier tipo de carta normalizada.

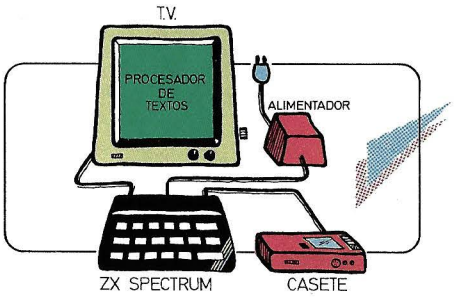
Hasta ahora tan sólo hemos empleado la opción 2 del menú principal para crear los documentos primario y secundario. Des-

El tratamiento de texto en equipos domésticos

El procesador de textos es una de las herramientas básicas para todo usuario de ordenadores personales. Incluso en los más pequeños de los microprocesadores (los denominados nano-ordenadores) es posible instalar versiones simplificadas de este tipo de programas. Por supuesto, sus prestaciones van a estar muy limitadas; tanto por el espacio de la memoria principal disponible (48 ó 64 kbytes, habitualmente), como por el volumen y técnicas de acceso a la memoria auxiliar (generalmente, cintas convencionales de tipo casete). Por ejemplo, con un procesador de texto destinado a un equipo doméstico, suele resultar imposible utilizar funciones como la de fusión descrita en este mismo artículo.

En todo caso, bien es cierto que los programas para el tratamiento de textos en nano-ordenadores facilitan las labores de escritura no excesivamente sofisticadas. Un ejemplo claro lo constituye la elaboración de informes de tamaño medio, que pueden ser perfectamente confeccionados con un procesador de textos de esta categoría. Las características generales de estos programas son las siguientes:

- Al utilizar como monitor un televisor normal, la resolución y capacidad de una línea del documento suele ser muy reducida (de 32 a 48 caracteres).
- Las posibilidades de gestión del texto (copias, modificaciones, etc.) salen fuera del ámbito del procesador de texto y, normalmente, deben ser realizadas mediante los comandos del sistema operativo.
- La producción de un documento escrito suele ser excluyente, es decir: mientras la impresora escribe el texto, el ordenador no puede ser utilizado en ninguna otra tarea.
- La paginación y numeración de páginas suele quedar a expensas del usuario, quien debe realizarla manualmente. Este debe reservar las líneas en blanco del principio y final de página, y teclear los números de orden para la paginación.
- Los procesos de sustitución automática de palabras no acostumbran a ser contemplados por los procesadores de texto de este nivel. De ahí que, habitualmente, deba realizarlos el propio usuario de forma manual.



En resumen, es importante tener en cuenta que estos programas resultan mucho más limitados que los procesadores de textos propiamente dichos, aunque también es importante citar que resultan útiles para múltiples tareas.

pués de su generación, se pueden utilizar las restantes opciones del menú para procesar por separado cada uno de los citados documentos. Por ejemplo, podríamos volver a editarlos para realizar modificaciones con la opción 1 "EDITAR UN DOCUMENTO ANTIGUO", o también, mediante la opción 3 "IMPRIMIR UN DOCUMENTO", escribirlo por la impresora, etc.

IMPRIMIR EL DOCUMENTO FUSION

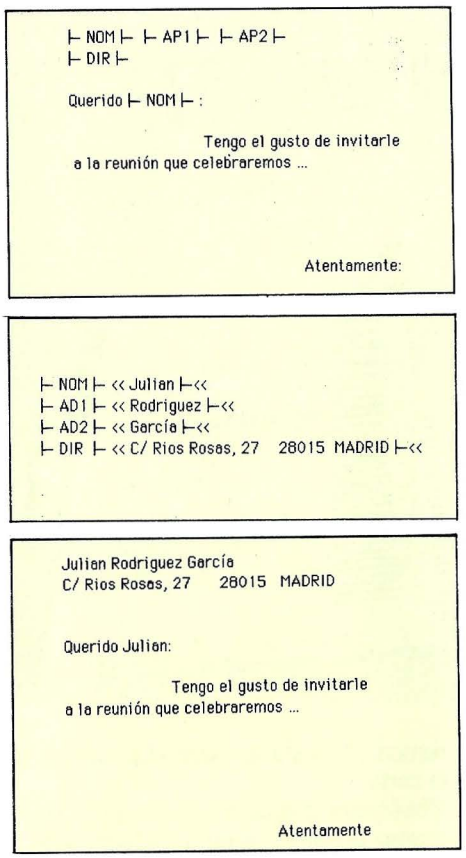
Después de haber creado y manipulado los documentos básicos, necesarios para la sesión de trabajo, se procederá directamente a seleccionar la opción 5, "IMPRIMIR UN DOCUMENTO FUSION", del menú principal. De inmediato aparecerá una pantalla en la que el programa MULTITEXTO solicita los nombres de los documentos a fusionar. En dicha pantalla habrá que especificar "CONVOCATORIA" como documento primario y "CLIENTES" como documento secundario; en este estricto orden, ya que el MULTITEXTO reconoce las características de los documentos a fusionar de acuerdo al orden en el que se le indican: el primero es el documento fijo y el segundo es el documento variable.

Después de especificar los nombres de los documentos a fusionar, bastará con pulsar la tecla RETURN para que el programa se encargue de producir tantas cartas de invitación como clientes existan en el documento secundario.

Sin lugar a dudas, la posibilidad de mezclar dos o más ficheros en un único documento de salida, es una de las facultades más importantes que pueden caracterizar a un programa para el tratamiento de textos. Precisamente, su presencia o ausencia nos permitirá distinguir los paquetes de calidad, de los programas que no pasan de ser imitaciones de un auténtico procesador de textos.

COMANDOS DEL MULTITEXTO PARA FUSION

Hasta ahora nos hemos limitado a describir las principales operaciones a efectuar



Aspecto de los documentos primario, secundario y de fusión. En primer lugar aparece el documento CONVOCATORIA, que contiene el esqueleto de la carta a enviar. En él se han reservado una serie de huecos a los que asigna un nombre. Estos nombres harán que los huecos tomen un contenido específico al fusionarse con el documento secundario.
La estructura del documento secundario es repetitiva, puesto que debe contener el mismo conjunto de variables particularizado para cada cliente.
El aspecto final del documento fusión muestra ya los huecos cumplimentados con los datos de cada uno de los clientes.

para producir la fusión de dos documentos mediante la opción 5 del MULTITEXTO. Para profundizar en las posibilidades que ofrece el programa en este tipo de operaciones, se definen a continuación los dos comandos más significativos que pueden ser utilizados por el usuario dentro de la función de FUSION:

- OB

Si en el punto de fusión del documento primario se añade el comando OB, el programa despreciará este apartado, sin dejar ningún hueco, en el caso de que para ese punto no exista información en el documento secundario.

- REPETIR

Cuando en el documento primario hay que repetir un bloque específico de variables, puede hacerse uso del comando REPETIR. Con éste se puede indicar la posición y el número de bloques que aparecerá en cada página.

RECAPITULACION

A lo largo de los tres últimos capítulos se ha analizado el programa para el tratamiento de textos MULTITEXTO. Sus características básicas deben haber quedado patentes a lo largo del estudio; no obstante, cabe terminar enunciando las principales ventajas e inconvenientes de este producto, que ayudarán a compararlo con otros paquetes similares:

Ventajas

1. Buena traducción al castellano tanto del programa como de la documentación.
2. El programa es rápido en su ejecución, con lo que el usuario se libera de dilatadas esperas para ver cumplidos sus mandatos.
3. Incorpora un diccionario ortográfico (en inglés) que permite revisar las faltas en un documento.
4. Buen sistema de pantallas y menús que sin ser excesivos facilitan el trabajo del usuario, permitiendo que una persona no especializada pueda utilizarlo cómodamente.
5. Facilidad para prefijar parámetros generales válidos para todos los documentos, lo que evita la necesidad de reiterarlos para cada uno de ellos.

Inconvenientes

1. Ausencia de opción para correo electrónico, con lo que el sistema sólo puede comunicarse con el usuario, nunca con otros sistemas.
2. Procedimiento algo complicado para el uso de impresoras no contempladas entre el conjunto de modelos que vienen prefijados en el programa base.
3. Imposibilidad de obtener copias de seguridad del programa; aunque en el fondo ello no hay que considerarlo un inconveniente, ya que los autores del programa tienen perfecto derecho a proteger sus intereses.

Un día en las carreras

Un sencillo programa para repasar conceptos

Toda la teoría explicada hasta ahora quedaría en "agua de borrajas" si no cruzara el umbral de su puesta en práctica. En el caso de la programación, la práctica se concreta en la creación de programas. Al respecto, no cabe duda de que la codificación de programas de tipo lúdico hace más entretenida la programación. Al tiempo, permite un estudio más ameno de los comandos del BASIC; estudio que resulta tan eficaz como el que derivaría de codificar otro tipo de programa más "serio".

Al igual que ha ocurrido en otros capítulos, el presente dedica sus páginas al desarrollo de ejemplos concebidos para ilustrar la puesta en práctica de la programación en lenguaje BASIC.

En este caso se trata de elaborar un programa de apuestas hípicas. Las galopadas de los caballos de carreras estarán presentes en el "hipódromo de la pantalla", logrando así una completa ambientación. El caballo ganador de la contienda se dejará a elección del azar. De esta forma, el resultado será totalmente imprevisible para el apostante. Este podrá, si tiene suerte, hacerse millonario (imaginariamente, claro), o bien sumirse en la más deplorable de las ruinas.

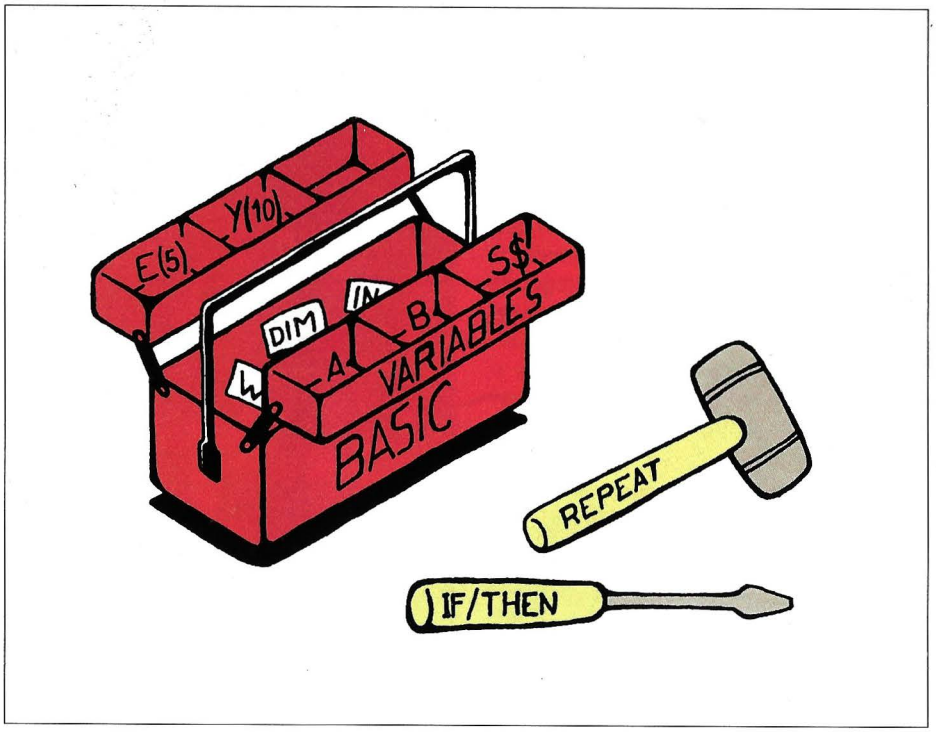
Para ir calentando "grupas", se comenzará con el diseño del cartel de entrada. Este servirá, además, para dar a conocer las reglas del juego. Su confección no tiene mayor dificultad que la de tratar de darle una presentación atractiva y, por ende, más eficaz. Este cartel de entrada puede realizarse por medio de la siguiente rutina:

```
10 REM TURF
20 CLS
30 PRINT AT 14,1 ;"***TURF***"
40 PRINT AT 0,6 ;"DISPONE INICIALMENTE
   DE 4.000 PTS."
50 PRINT :PRINT "LA APUESTA MAXIMA
   ES DE 1.500 PTS."
60 PRINT :PRINT "SI GANA EL CABALLO
   APOSTADO, SE"
70 PRINT "MULTIPLICARA POR TRES LA
   CANTIDAD JUGADA"
80 PRINT AT 2,23; "PULSE UNA TECLA PARA
   COMENZAR";
90 LET B$=INKEY$ :IF B$=""THEN GOTO 90
```

A la hora de confeccionar este programa, se ha supuesto que la presentación en pantalla es de 24 filas (de la 0 a la 23) por 40 columnas (de la 0 a la 39). El resultado que se obtendrá coincidirá con el siguiente:

```
        *** TURF ***

DISPONE INICIALMENTE DE 4.000 PTAS.

LA APUESTA MAXIMA ES DE 1.500 PTAS.

SI GANA EL CABALLO APOSTADO, SE
MULTIPLICARA POR 3 LA CANTIDAD
JUGADA

PULSE UNA TECLA PARA COMENZAR
```

En el caso particular de que el ordenador utilizado no comparta dicho tamaño de pantalla, habría que acomodar el cartel de



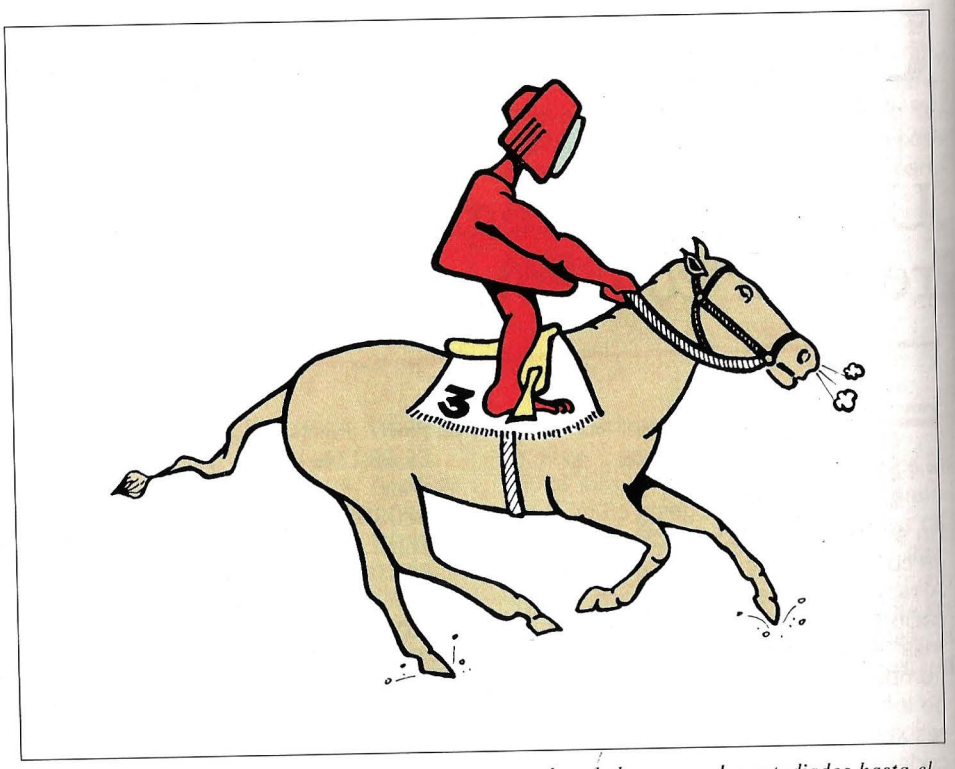
El BASIC es un lenguaje de programación potente, versátil y repleto de útiles herramientas para el diálogo con el ordenador. La práctica de la programación es el camino a seguir para llegar a utilizarlo con eficacia y sacar un buen partido a sus posibilidades.

entrada al número de filas y columnas disponibles. Esto mismo afectará también a la presentación en pantalla del resto del programa. Por ello es aconsejable consultar previamente el manual que acompaña a cada equipo. Utilizando los comandos PRINT y PRINT AT, se sitúan los diversos textos en su lugar correspondiente, para que queden centrados y produzcan una presentación agradable.

Al final del segmento de programa que se encarga de crear el cartel de entrada, se encuentra una instrucción que hace uso de la función INKEY$ y una sentencia IF/THEN (línea 90). A primera vista, la referida línea de programa parece totalmente ajena al objetivo del cartel de entrada. Su cometido no es más que el de producir una espera por parte del ordenador, para dar tiempo a leer el cartel de entrada. De otra forma, el programa seguiría ejecutando las líneas que se incluirán a continuación y el cartel desaparecería casi instantáneamente sin cumplir su cometido.

Para evitarlo, mediante el comando INKEY$ se comprueba si se ha pulsado alguna tecla. Si es así, proseguirá la ejecución del programa en la línea siguiente (que aún no ha sido escrita). De lo contrario, se volverá a comprobar si se ha pulsado ya una tecla (GOTO 90). Así pues, el segmento principal del programa no empezará a ejecutarse hasta que lo ordene el usuario; orden que introducirá pulsando una tecla cualquiera.

El resto del programa se puede dividir en tres segmentos con entidad propia. Una primera parte indicará la cantidad de dinero de la que se dispone, y pedirá la que se desea apostar, así como el caballo al que va dirigida la apuesta. La segunda parte se encargará de simular la carrera de caballos a lo largo de la pantalla. Por último, en la tercera parte se comprobará si el caballo ganador coincide con el apostado, para así actualizar la "bolsa" del apostante.



A lo largo de este capítulo se llevan a la práctica muchos de los comandos estudiados hasta el momento, con el fin de confeccionar un programa capaz de trasladar al ordenador toda la emoción de una carrera de caballos.

HAGAN JUEGO, SEÑORES...

La primera parte del núcleo del programa se ocupará, como ya se ha indicado, de

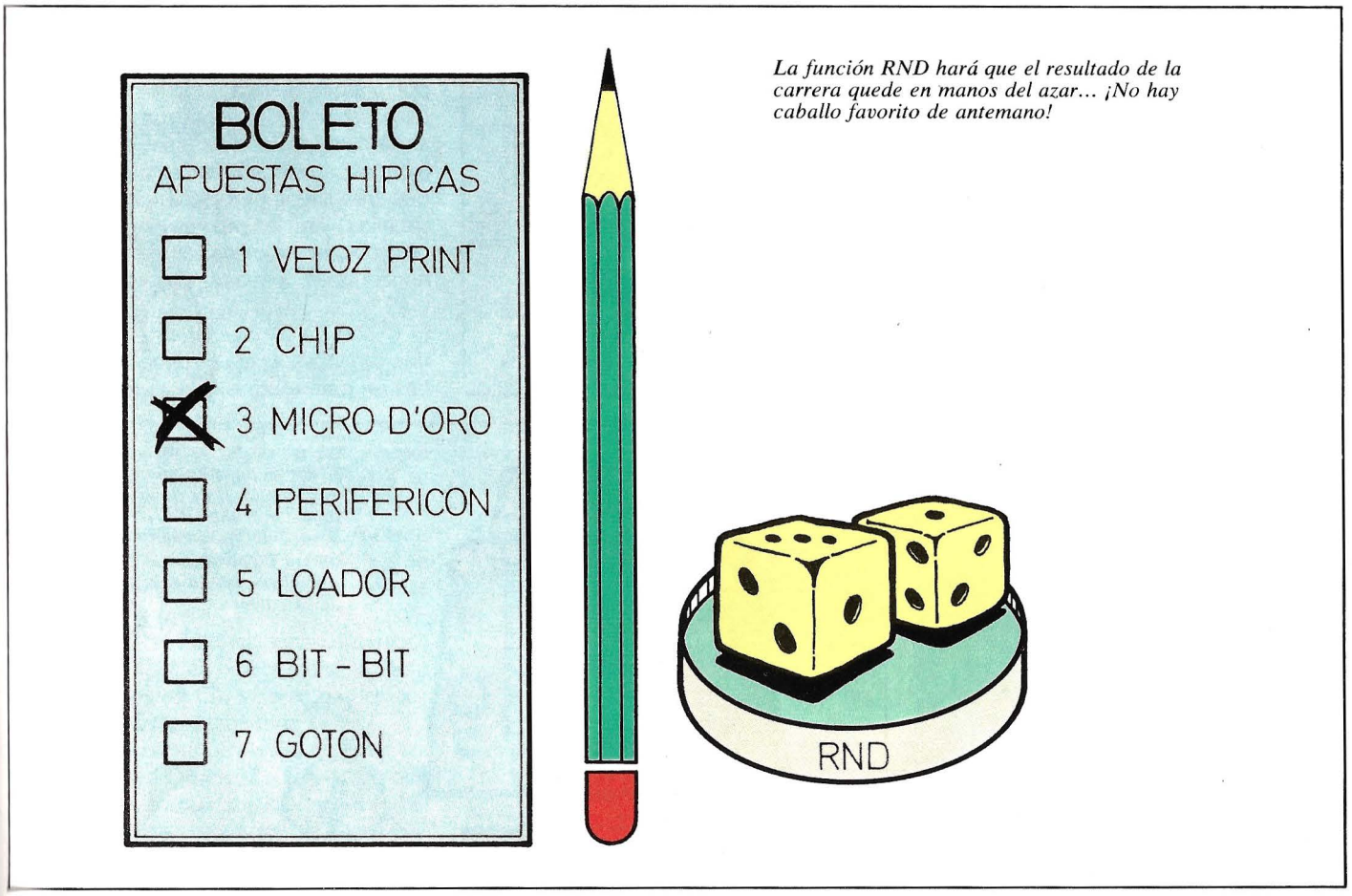
El BASIC no es más que un lenguaje destinado a facilitar la comunicación; desde luego, su diferencia respecto a los lenguajes humanos reside en que su objetivo es permitir el diálogo con las máquinas programables: los ordenadores.
Su práctica puede resultar amena, e incluso divertida, tomando como elemento de ejercicio la confección de un programa de tipo lúdico.

recoger la cantidad a apostar y el número del caballo en el que se va a invertir. Pero antes de ello habrá que inicializar una variable (D) con la cantidad de dinero disponible para apuestas. Esta será, como se indicó en el cartel de entrada, de 4.000 ptas. Dicha acción se efectúa en la línea 100 que contiene, además, el dimensionado de las matrices X y C; matrices que se utilizarán más adelante para memorizar las coordenadas actuales de cada uno de los caballos en carrera.
El listado de esta primera parte es el siguiente:

```
100 DIM X(5):DIM C(5):LET D=4000
110 CLS:IF D=0 THEN PRINT "NO QUEDA
    DINERO PARA APOSTAR":END
120 PRINT "TIENE ";D;" PTS."
130 PRINT
140 INPUT "CUANTO APUESTA ";A
150 IF A>0 AND A<=1500 AND A<=D THEN
    GOTO 180
160 PRINT "APUESTA NO VALIDA"
170 GOTO 130
180 PRINT
190 INPUT "(DEL 1 AL 5) A QUE CABALLO ";C
200 IF C<1 OR C>5 THEN GOTO 180
210 LET D=D-A
```

La línea 110 borra la pantalla con CLS, para dejarla en condiciones de ser reutilizada en esta primera zona del programa. De esta forma, nada habrá que estorbe o distraiga la concentración del apostante.
Como quiera que esta zona debe ejecutarse de nuevo cada vez que concluya una carrera, sea cual fuere el resultado de la misma, se ha incluido en la propia línea 110 una comprobación de la cantidad que posee el apostante. El programa finalizará si de esta comprobación resulta que la cantidad de dinero disponible es igual a cero... ¡el jugador se ha arruinado!
La línea 120 indicará en cada pasada la cantidad actual de que dispone el apostante. Más adelante, la línea 140 pide, mediante un comando INPUT, la cantidad que se va a apostar en la próxima carrera; cantidad que se almacenará en la variable A.

En la línea siguiente se comprueba si la cantidad es correcta. Para ello debe tratarse de un número mayor que cero (A>0), menor o igual que la apuesta máxima, que es de 1.500 ptas. (A≤1.500) y menor o igual que la cantidad que posee actualmente el apostante (A≤D). Esta comprobación se realiza agrupando las condiciones antes citadas en una sola, entorno al operador lógico AND. Así, pues, deben cumplirse todas ellas para que esta nueva expresión lógica sea cierta (valor lógico 1). La mencionada expresión se utiliza para decidir, dentro de la instrucción IF/THEN de la línea 150, qué acción hay que emprender: saltar a la línea 180 del programa, si es cierta la expresión, o bien presentar el mensaje de "APUESTA NO VALIDA" y volver a la línea 130 para pedir una nueva apuesta. Esto último, en el caso de que la expresión adoptara el valor lógico 0 (falso). De forma totalmente análoga, se pedirá el número del caballo al que se dirige la apuesta. Dicho valor se asignará a la variable C, en la línea 190. A continuación se comprueba, en la línea



La función RND hará que el resultado de la carrera quede en manos del azar... ¡No hay caballo favorito de antemano!

200, si el valor introducido es correcto. En este caso se emplea el operador lógico OR para construir la expresión de control de la instrucción IF/THEN. Esta adoptará el valor lógico 1 cuando C sea menor que 1, o cuando sea mayor que 5 (caballos no válidos), siendo entonces cierta la condición y ejecutándose la zona THEN. Así se envía el flujo de ejecución del programa a la línea 180, en la que se pide de nuevo un valor para C, ya que el anterior no era válido.

Finalmente, en la línea 210 se resta del total, contenido en la variable D, la cantidad apostada. Las líneas situadas a continuación corresponderán a la segunda parte del programa. Zona que se encargará de realizar la carrera propiamente dicha.

El resultado en pantalla de una posible ejecución de esta primera parte sería el siguiente:

```
TIENE 4000 PTS.

CUANTO APUESTA ?2000

APUESTA NO VALIDA

CUANTO APUESTA ?100

(DEL 1 AL 5) A QUE CABALLO ?3
■
```

NO VA MAS...

La segunda zona del programa debe simular en la pantalla la carrera de los cinco veloces "purasangre". A modo de recordatorio, se indicará la cantidad y el número del caballo apostados; desde luego, después de haber utilizado el comando CLS para dejar limpia la "pista de carreras".

Acto seguido se trazará la línea de salida y la línea de llegada. Para ello se hace uso de la sentencia repetitiva FOR/NEXT, junto con los correspondientes PRINT AT, que imprimirá un signo consistente en una barra vertical (o el signo más parecido de que se disponga, para lo que habrá que consultar el manual del equipo utilizado y buscar en el repertorio de caracteres).

Queda aún una tarea previa, antes de pasar a codificar la zona que se ocupará de "abrir los cajones de los caballos": inicializar las variables para las coordenadas horizontal y vertical de cada caballo. Estas coordenadas se almacenarán en cada uno de los cinco elementos de las matrices X y C. Como se recordará, ambas fueron dimensionadas en la línea 100 del segmento anterior.

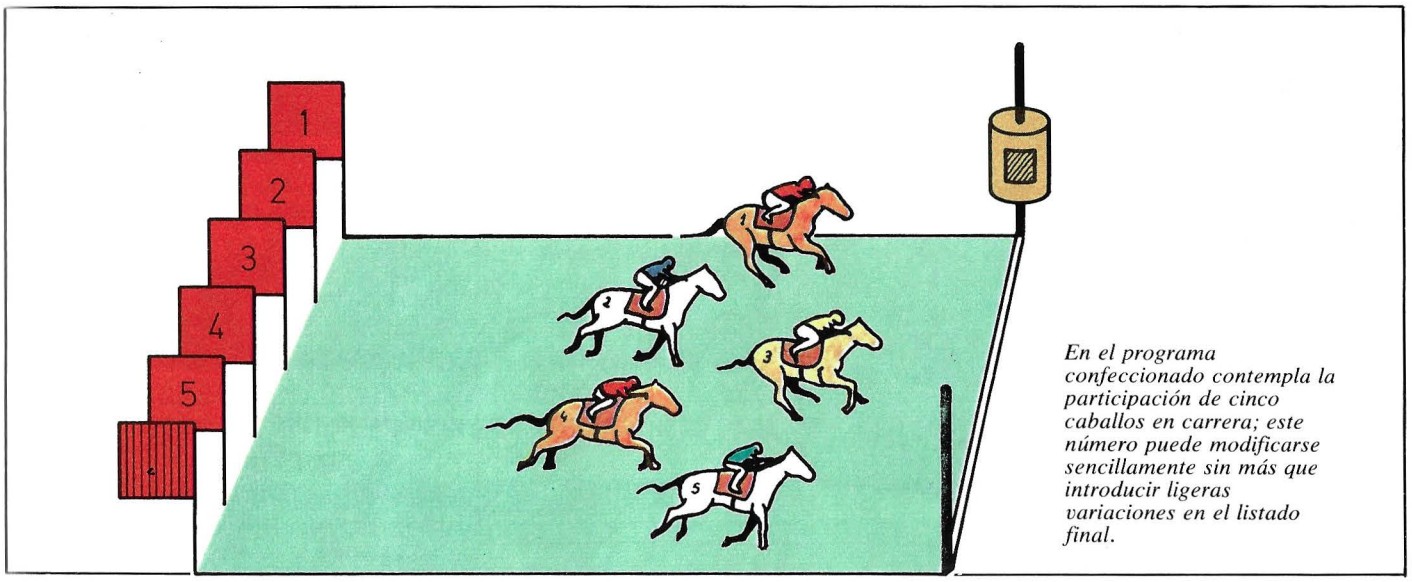
En la variable suscrita X() se almacenará el valor actual de la coordenada horizontal. Su contenido debe ser incrementado (de forma aleatoria), puesto que la carrera transcurrirá desde el margen izquierdo hasta el derecho de la pantalla. La matriz C(), en cambio, mantendrá siempre valores constantes, que corresponden a la coordenada vertical; valores que delimitan la línea o "calle" por la que avanzará cada caballo.

El contenido inicial de esta matriz marca, por lo tanto, la línea imaginaria que seguirá cada caballo. Para lograr una distribución simétrica y evitar que "se molesten unos a otros", se elegirán de tal forma que queden tres líneas en blanco entre cada dos caballos. Así, los valores elegidos, siempre considerando una pantalla de 24 líneas horizontales, pueden ser: 3, 7, 11, 15 y 19.

El hecho de almacenar estos valores en una variable suscrita se debe a que simplifica en gran medida la zona del programa

La pantalla del ordenador se convierte en esta ocasión en un improvisado hipódromo gracias a la magia del BASIC.

El juego permite apostar una cantidad de dinero cuyo límite máximo se determina dentro del programa; dinero que el jugador puede perder o triplicar según el resultado de la carrera.



En el programa confeccionado contempla la participación de cinco caballos en carrera; este número puede modificarse sencillamente sin más que introducir ligeras variaciones en el listado final.

que se encargará de situar a cada caballo en su lugar correspondiente. Así, se puede incluir esta acción en un bucle FOR/NEXT. De otra forma se necesitaría una línea de programa para cada uno de los cinco caballos. Ello haría quizás más rápida la ejecución del programa, pero, evidentemente, el listado sería más extenso. Este segmento del programa adopta el siguiente aspecto:

```
220 CLS
230 PRINT AT 4,0;"APOSTADAS ";A;" PTS.
    AL CABALLO ";C
240 FOR L=1 TO 21:PRINT AT 2,L;"|":PRINT
    AT 37,L;"|":NEXT L
250 LET C(1)=3:LET C(2)=7:LET C(3)=11:LET
    C(4)=15:LET C(5)=19
260 FOR I=1 TO 5:LET X(I)=0:LET H=C(I):
    GOSUB 500:NEXT I
```

Como ya se indicó, en dicha porción del programa se muestran la cantidad y el caballo que son objeto de la apuesta. Acción que se realiza en la línea 230. A continuación (línea 240) se trazan los límites de la pista por medio del bucle anteriormente explicado.

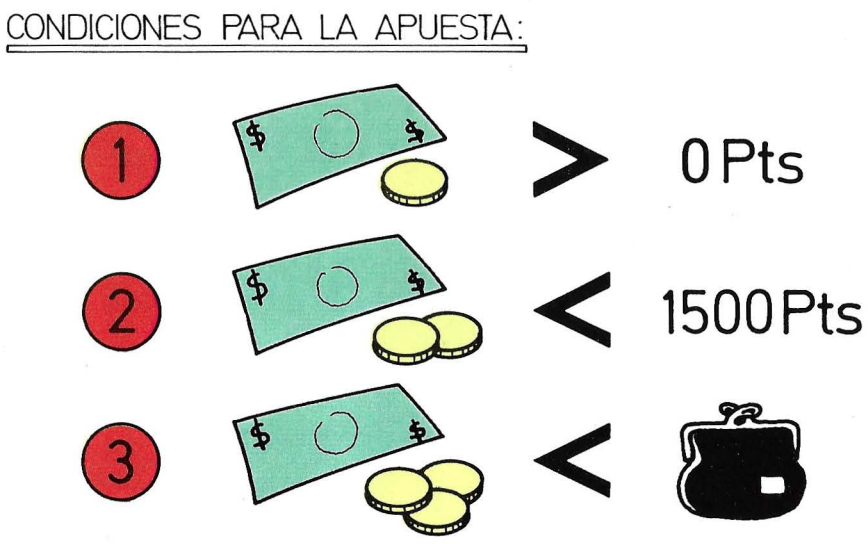
El objeto de las instrucciones contenidas en la línea 250 no es otro que dar valores a los elementos de C(); valores que coinciden con los propuestos más arriba.

La línea 260 utiliza de nuevo una instrucción iterativa FOR/NEXT. Esta se repite tantas veces como caballos corren (5). El mismo bucle se aprovecha para inicializar la matriz X con ceros (LET X(I)=0), que son las coordenadas horizontales de partida de los caballos. Dentro del bucle se llama a una subrutina que se situará a partir de la línea 500. La misión de esta subrutina es la de mostrar en pantalla cada caballo en su posición actual. Justo antes de invocar a la subrutina, se almacena en la variable H la coordenada vertical del caballo en curso (recuérdese que en cada pasada la variable I toma un valor diferente, este valor sirve para identificar a cada caballo).

La referida subrutina aparece codificada en las siguientes líneas:

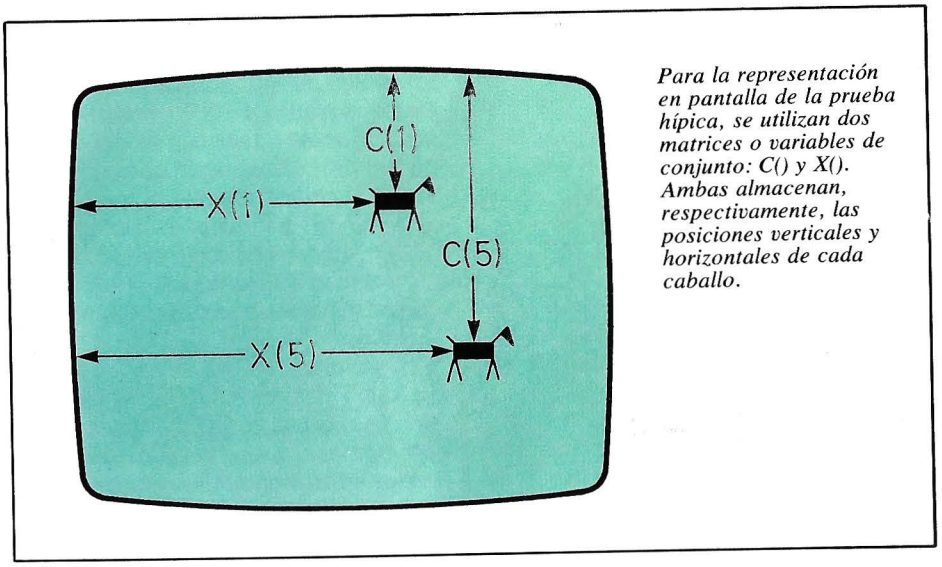
```
500 LET M=X(1)*(H=C(1))+X(2)*(H=C(2))+
    X(3)*(H=C(3))+X(4)*(H=C(4))+X(5)*
    (H=C(5))
510 PRINT AT M,H;"<";(H+1)/4;">"
520 RETURN
```

Su funcionamiento es el siguiente: mediante un comando PRINT AT se selecciona el punto de la pantalla en el que se




```
IF A>0 AND A<= 1500 AND <= D THEN GO TO 180
```

En el programa se comprueba la validez de la apuesta examinando el contenido de la variable A. Las tres condiciones impuestas se funden en una sola expresión por medio del operador lógico AND.



Para la representación en pantalla de la prueba hípica, se utilizan dos matrices o variables de conjunto: C() y X(). Ambas almacenan, respectivamente, las posiciones verticales y horizontales de cada caballo.

va a situar al caballo. Para ello, la coordenada horizontal se calcula mediante la expresión de la línea 500:

```
M=X(1)*(H=C(1))+X(2)*(H=C(2))+X(3)*(H=C
  (3))+X(4)*(H=C(4))+X(5)*(H=C(5))
```

Esta expresión se puede dividir para su estudio en dos partes: una lógica y otra aritmética. La parte lógica calcula el valor lógico (0 ó 1) de los términos del tipo H=C(j), siendo j un valor comprendido entre 1 y 5. Sólo una de las cinco expresiones será cierta: aquella cuyo valor actual de H coincida con el valor correspondiente de la matriz C(). Por lo tanto, ésta adoptará el valor lógico 1, mientras que las demás tomarán el valor 0. La zona aritmética se limita a multiplicar ese valor lógico por el valor actual de la coordenada horizontal. Esta última se encuentra almacenada en la matriz X(). Así, por ejemplo, si al efectuar la llamada a la subrutina se asigna a la variable auxiliar H el valor contenido en C(2), sólo el paréntesis que contiene a la expresión H=C(2) adoptará el valor 1. Todos los demás valdrán 0, con lo que el producto de ese 1 por X(2) dará como resultado el valor contenido en X(2), siendo nulos los demás sumandos.

La otra coordenada corresponde simplemente al valor de H. Una vez conocida la posición, se imprime en pantalla el símbolo del caballo. Para ello se hace uso del número de cada animal. Este se calcula a partir del valor de la variable H. La relación entre H (posición vertical del caballo en curso) y el número identificativo del caballo viene dada por la fórmula:

```
<número>=(H+1)/4
```

Esta fórmula habrá de ser corregida en el caso de elegir unas coordenadas verticales distintas a las aquí utilizadas.

En este ejemplo se ha sacrificado la estética en función de la estandarización. Por ese motivo los caballos se representarán simplemente por medio de su número, escoltado por los símbolos "<" y ">". Justo delante de estos tres símbolos ("<", número del caballo y ">") se imprimirá un espacio en blanco. Este tiene como misión borrar la estela que iría dejando cada caballo al avanzar por la pista, debido a la impresión sucesiva de los símbolos "<1>".

Una vez introducido este nuevo fragmento, la pantalla mostrará al ejecutarlo una imagen similar a la siguiente:

```
APOSTADA 100 PTS. AL CABALLO 3

<1>

<2>

<3>

<4>

<5>
```

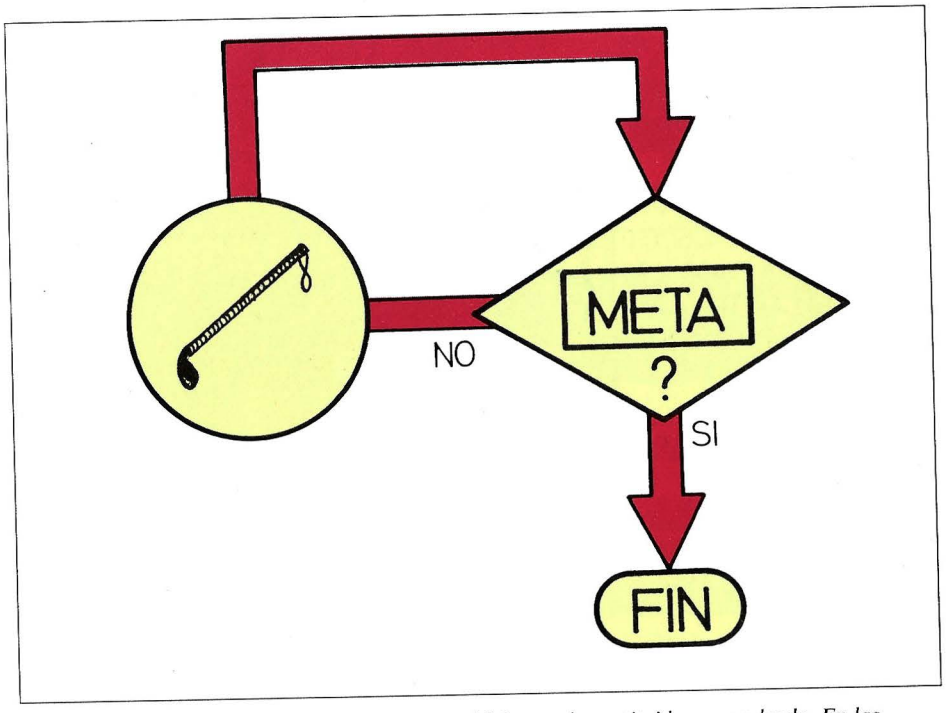
El programa que aparece en el listado se ha confeccionado utilizando un dialecto BASIC genérico. Ello significa que, antes de introducirlo en su ordenador, debe examinar las diversas instrucciones y acondicionar su formulación —si ello fuera necesario— a la versión específica del lenguaje BASIC que utilice su equipo. Las tablas de conversión incluidas en los capítulos precedentes pueden constituir una eficaz ayuda para realizar tal adaptación.

```
10 REM TURF
20 CLS
30 PRINT AT 14,1 ; "*** TURF ***"
40 PRINT AT 0,6 ; "DISPONE INICIALMENTE DE 4.000 PTS."
50 PRINT : PRINT "LA APUESTA MAXIMA ES DE 1.500 PTS."
60 PRINT : PRINT "SI GANA EL CABALLO APOSTADO, SE"
70 PRINT "MULTIPLICARA POR TRES LA CANTIDAD JUGADA"
80 PRINT AT 2,23; "PULSE UNA TECLA PARA COMENZAR"
90 LET B$ = INKEY$ : IF B$ = "" THEN GOTO 90
100 DIM X(5):DIM C(5):LET D=4000
110 CLS:IF D=0 THEN PRINT "NO QUEDA DINERO PARA APOSTAR":END
120 PRINT "TIENE ";D;" PTS."
130 PRINT
140 INPUT "CUANTO APUESTA ";A
150 IF A>0 AND A<=1500 AND A<=D THEN GOTO 180
160 PRINT "APUESTA NO VALIDA"
170 GOTO 130
180 PRINT
190 INPUT "(DEL 1 AL 5) A QUE CABALLO ";C
200 IF C<1 OR C>5 THEN GOTO 180
210 LET D=D-A
220 CLS
230 PRINT AT 4,0;"APOSTADAS ";A;" PTS. AL CABALLO ";C
240 FOR L=1 TO 21:PRINT AT 2,L;"|":PRINT AT 37,L;"|":NEXT L
250 LET C(1)=3:LET C(2)=7:LET C(3)=11:LET C(4)=15:LET C(5)=19
260 FOR I=1 TO 5:LET X(I)=0:LET H=C(I):GOSUB 500:NEXT I
270 FOR J=1 TO 5
280 LET H=C(J):LET X(J)=X(J)+INT(RND+0.5):GOSUB 500
290 IF X(J)>33 THEN LET J=6:NEXT J:GOSUB 600:GOTO 110
300 NEXT J:GOTO 270
500  LET  M=X(1)*(H=C(1))+X(2)*(H=C(2))+X(3)*(H=C(3))+X(4)*(H=C(4))+X(5)*
(H=C(5))
510 PRINT AT M,H;" <";(H+1)/4;">"
520 RETURN
600 IF C=(H+1)/4 THEN PRINT AT 3,23;"ENHORABUENA, HA GANADO ";A*3;" PTS.":LET
D=D+A*3:GOTO 620
610 PRINT AT 3,23 "LO SIENTO, GANO EL CABALLO ";(H+1)/4
620 FOR E=1 TO1000:NEXT E:RETURN
```

Llegado el programa a este punto, tan sólo queda, para poner fin a esta segunda parte, la zona encargada de hacer correr a los caballos y de detectar la llegada del ganador a la meta. El listado de las líneas que corresponden a dicha zona es el que figura a continuación.

```
270 FOR J=1 TO 5
280 LET H=C(J):LET X(J)=X(J)+INT(RND+
    0.5):GOSUB 500
290 IF X(J)>33 THEN LET J=6:NEXT J:GOSUB
    600:GOTO 110
300 NEXT J:GOTO 270
```

Todo este segmento está incluido en un par de bucles. El primero de ellos (creado por el GOTO 270 de la línea 300), se encarga de repetir todo el bloque un número de veces suficientemente grande (teóricamente, infinitas veces). Con ello se logra tener la certeza de que alguno de los ca-



La zona del programa que controla la carrera hípica está constituida por un bucle. En las sucesivas ejecuciones del mismo, se avanzan aleatoriamente las posiciones de los distintos caballos en carrera, hasta que uno de ellos alcanza la meta.

ballos llega a la meta. La llegada se comprueba en la línea 290, simplemente examinado si la coordenada horizontal de alguno de los caballos es superior a 33. Si es así, se dará por finalizado el bucle interno, poniendo su contador (J) al máximo de la cuenta.

Seguidamente se verifica si el caballo ganador coincide con el apostado. Esto último se ha de realizar en la subrutina colocada a partir de la línea 600. Más tarde se regresa a la línea 110 (GOTO 110) para que tenga lugar una nueva apuesta.

El bucle más interno (formado por los comandos FOR y NEXT de las líneas 270 y 300) se repetirá una vez por cada caballo, siendo la línea 280 la que hará avanzar aleatoriamente a los equipos. Ello se efectúa mediante el comando RND, al que se añade la cantidad 0.5, de tal forma que el resultado de todo ello tendrá la misma probabilidad de ser 0 ó 1.

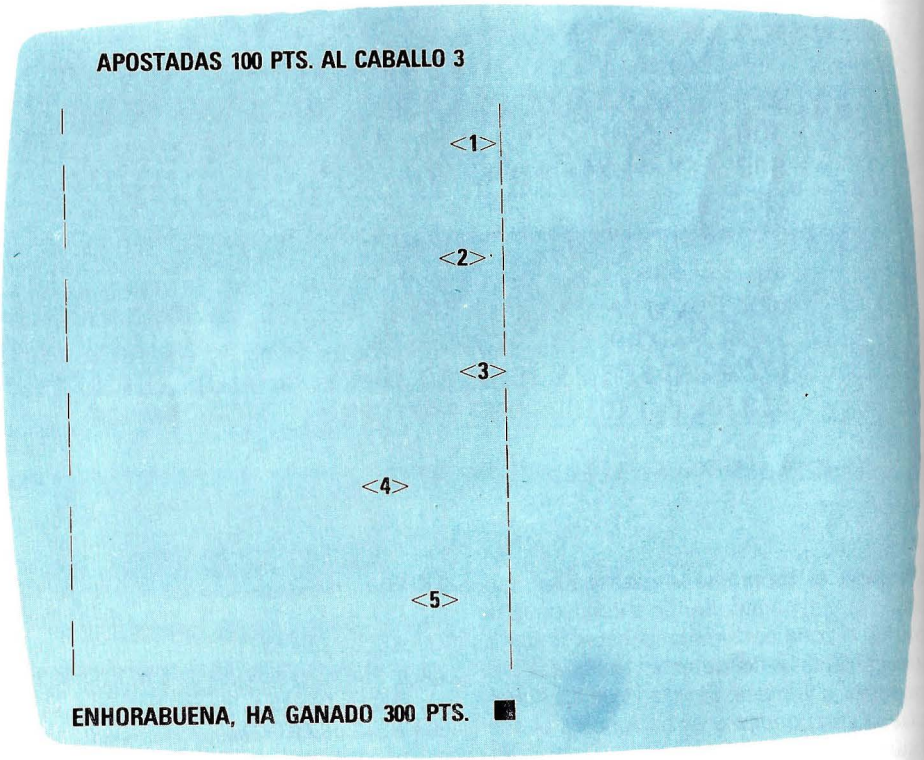
El resultado de dicha operación es sumado al antiguo valor de la coordenadora horizontal. Una llamada a la subrutina de la línea 500 sitúa al caballo correspondiente en el nuevo punto de la pantalla.

La salida de este segmento de programa tiene lugar en la línea 290, mediante la llamada a una nueva subrutina situada a partir de la línea 600. Subrutina que constituye lo que hasta ahora se ha venido denominando "tercera parte del programa" y cuyo listado se ofrece a continuación:

```
600 IF C=(H+1)/4 THEN PRINT AT 3,23;
    "ENHORABUENA, HA GANADO ";A*3;"
    PTS.":LET D=D+A*3:GOTO 620
610 PRINT AT 3,23 "LO SIENTO, GANO EL
    CABALLO";(H+1)/4
620 FOR E=1 TO1000:NEXT E:RETURN
```

Esta última parte no tiene mayor complicación que la de comprobar si el caballo ganador coincide con el que recibió la apuesta. De darse esta circunstancia, se procederá a calcular el importe ganado, mediante una simple multiplicación por tres. En el caso de que el pronóstico no haya sido el acertado, se imprimirá el correspondiente mensaje.

Antes de devolver el control al punto de llamada de la subrutina, se establece un retardo mediante un nuevo bucle FOR/NEXT vacío. Retardo que permite ver el resultado. Una vez transcurrido ese tiempo se regresará al punto de partida. Desde allí se salta a la línea 110, donde dará comienzo una nueva apuesta. El programa há sido ya elaborado en su totalidad y ofrece el aspecto que refleja el listado adjunto. Como ejemplo del resultado obtenido, se muestra a continuación la pantalla final correspondiente a una hipotética ejecución:



Pascal (8)

Nuevos tipos de datos no estructurados

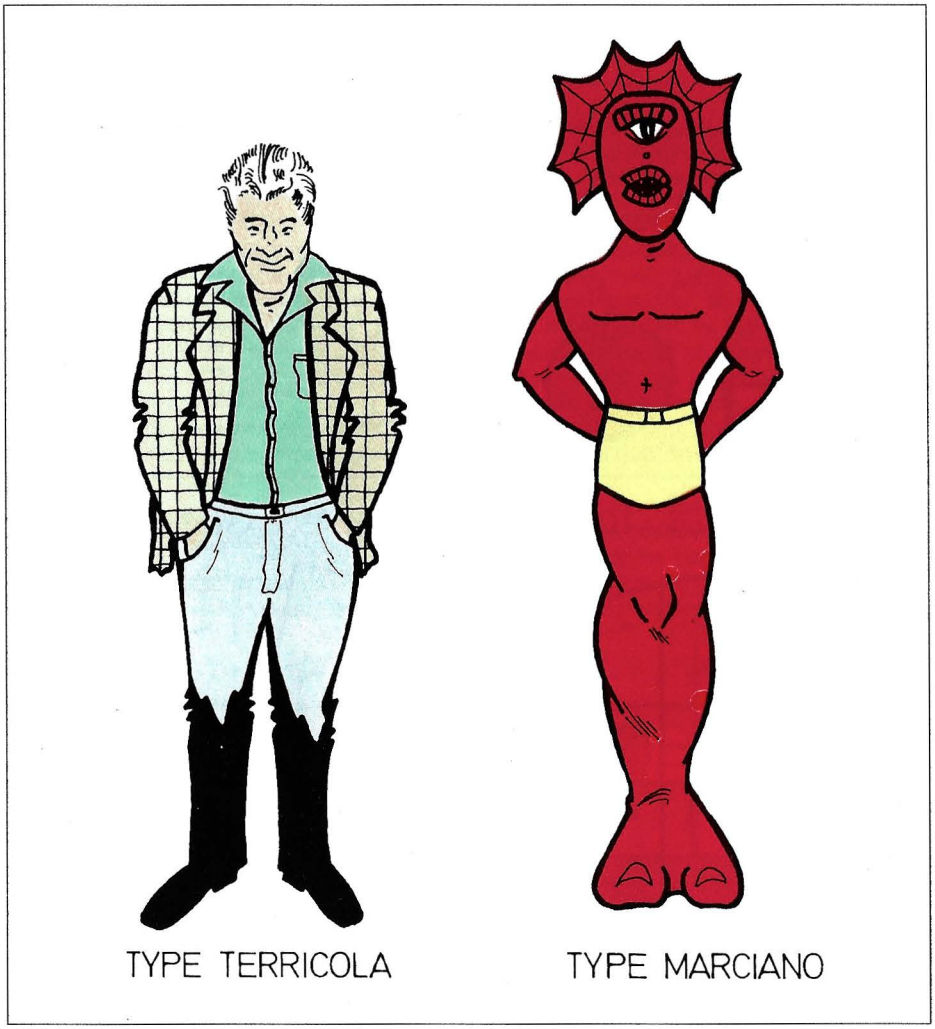
Para que sea posible utilizar cualquier elemento del PASCAL, ya se trate de constantes, variables, funciones o expresiones, es imprescindible su declaración previa dentro del programa. Esta declaración previa permite asignar a cada elemento del PASCAL lo que denominamos "tipo"; de forma que cada elemento será de un tipo determinado y sólo uno. El tipo de cada dato determinará el conjunto de posibles valores que éste puede adoptar, así como los operadores que se podrán utilizar con él. No hay que olvidar que cada tipo de datos admite un determinado conjunto de operadores que producirán resultados de ese mismo tipo.

El hecho de distinguir entre tipos de datos facilita la tarea de programación y agiliza la detección de errores. Al trabajar con lenguajes que no utilizan tipos de datos, es frecuente que se produzcan errores muy difíciles de detectar; por ejemplo, al intentar "sumar peras con manzanas" (cantidades ambas que estarán expresadas en forma de números enteros). Al utilizar diferentes tipos de datos para las peras y las manzanas, quedan obviados los errores de esta naturaleza.

En el capítulo precedente se vio cómo la declaración de las variables con VAR asigna instantáneamente el tipo especificado a dichas variables. Asimismo, mediante la declaración de constantes con CONST, era posible que una futura modificación de esos valores constantes sólo afectara a su declaración, y no a todas las sentencias en las que figurase esa constante. De forma similar, el PASCAL permite dar un nombre o identificador a un tipo de dato determinado mediante la llamada "definición de tipo". Ello contribuye a facilitar en mayor medida el manejo de los tipos de datos. El formato de esta definición de tipo es el siguiente:

TYPE <identificador> = <tipo>;

Toda definición de tipo debe preceder en el programa a la declaración de variables con VAR. Es importante reseñar que esta definición de tipo NO declara a una variable, sino únicamente a un tipo de dato con un nuevo nombre; nombre que se podrá utilizar posteriormente como tal en la declaración de variables o de otros elementos del PASCAL. En el siguiente ejemplo se observa la puesta en práctica de una definición de esta índole:

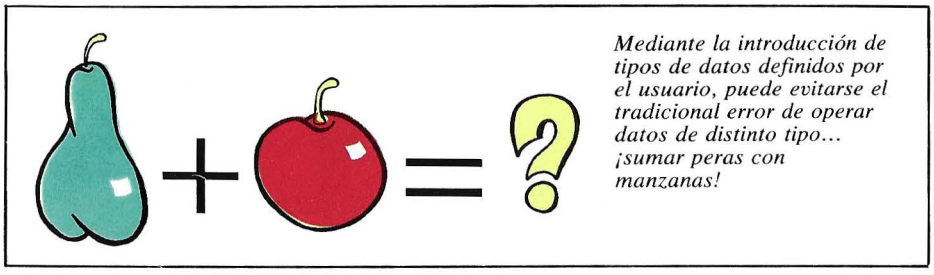


La posibilidad de definir distintos tipos de datos permite diferenciar a éstos entre sí. La propia definición de tipo delimita, al tiempo, los posibles valores que puede adoptar cada dato.

TABLA DE COMANDOS-PASCAL

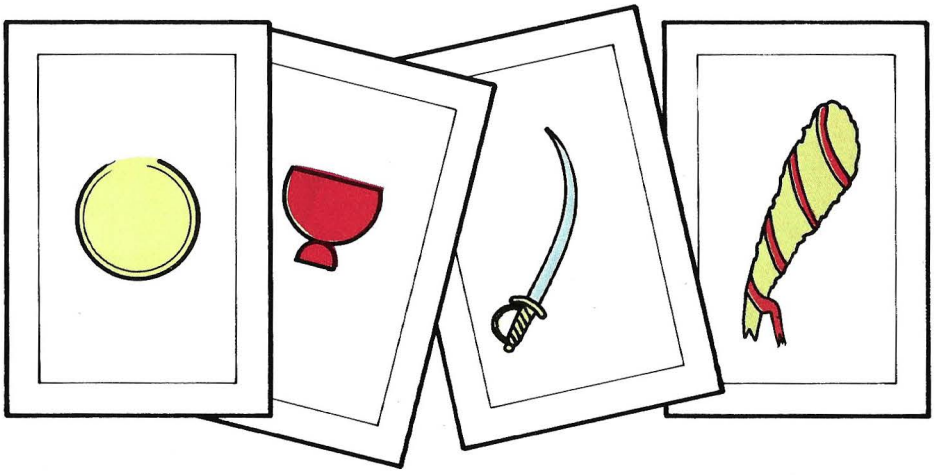
COMANDO	COMETIDO	OBSERVACIONES
TYPE <id.> = <tipo>;{...}	Declaración de tipos de datos no estándar.	Zona de declaraciones y definiciones previa a VAR.
TYPE <id.> = (<id.1>,..., <id.n>);	Declaración de tipos por enumeración de datos.	Zona de declaraciones y definiciones previa a VAR.
TYPE <id.> = <cte.>.. <cte.>;	Declaración de tipos subrango de datos.	Zona de declaraciones y definiciones previa a VAR.
PRED (X);	Función que calcula el predecesor de un dato ordinal	Cuerpo del programa.
SUCC (X);	Función que calcula el sucesor de un dato de tipo ordinal.	Cuerpo del programa.
ORD (X);	Función que calcula el número de orden de un dato de tipo ordinal.	Cuerpo del programa.
CHR (I);	Función que calcula el carácter correspondiente al número ordinal I.	Cuerpo del programa.

NOTA: <id.>: identificador válido PASCAL. <tipo>: cualquier tipo escalar de datos. <id.1>,...,<id.n>: identificadores de constantes de un tipo de datos. <cte.>: valor constante de un tipo de dato escalar excepto REAL. (X): dato de tipo ordinal. (I): número natural.



Mediante la introducción de tipos de datos definidos por el usuario, puede evitarse el tradicional error de operar datos de distinto tipo... ¡sumar peras con manzanas!

TYPE PALOS = (OROS, COPAS, ESPADAS, BASTOS);



La definición de datos del tipo enumerado se lleva a efecto relacionando, ordenadamente, cada uno de los nuevos valores a crear.

```
TYPE
        numero=REAL;
        respuesta=BOOLEAN;

VAR
        temperatura:numero;
        calor:respuesta;
```

Estas declaraciones equivalen a la siguiente:

```
VAR
        temperatura:REAL;
        calor:BOOLEAN;
```

TIPOS DE DATOS NO ESTRUCTURADOS

Tipos de datos no estructurados, o escalares estándar, son aquellos cuyos posibles valores no se pueden descomponer en otro tipo de dato. Estos tipos de escalares estándar, ya estudiados en anteriores capítulos de la obra, son: INTEGER, REAL, CHAR y BOOLEAN.

Los posibles valores de estos tipos de datos están ordenados, esto es: al evaluar dos datos, siempre se puede establecer un orden ascendente o descendente entre ellos. En virtud de esta característica, quedan definidos los operadores de relación (menor que, mayor o igual que...), así como la sentencia de asignación. Una expresión de cualquier tipo escalar, excepto el REAL, puede ser variable de control de un bucle FOR o selector de una sentencia CASE/OF.

No obstante, hay muchas ocasiones en las que el usuario desearía poseer otros tipos de datos, que le permitieran manejar con mayor facilidad una cierta información. Así, por ejemplo, en determinadas ocasiones, convendría utilizar un tipo de datos cuyos valores fuesen los nombres de los días de la semana, o los tres colores básicos; o incluso los palos de una baraja de cartas española. El PASCAL permite la definición de nuevos tipos de datos escalares por parte del usuario. Esta definición puede llevarse a efecto según dos métodos: por enumeración, o por definición de un subconjunto de cualquier otro tipo de dato escalar (tipos subrango).

EL TIPO ENUMERADO

La definición de tipo por enumeración se realiza especificando ordenadamente todos y cada uno de los valores del nuevo tipo que se quiere definir. Los valores se representarán por medio de identificadores; estos constituirán las constantes de ese tipo de datos definido, y se declararán mediante la siguiente definición:

TYPE <identificador del tipo>=(<id.1>, <id.2>,...,<id.n>);

El identificador del tipo será el nuevo nombre de ese tipo de datos, mientras que la lista encerrada entre paréntesis coincidirá con los posibles valores que pueden adoptar los elementos que se declaren. Veamos un ejemplo al respecto:

TYPE FRUTAS:(PERAS,MANZANAS, FRESAS);

VAR f:FRUTAS;

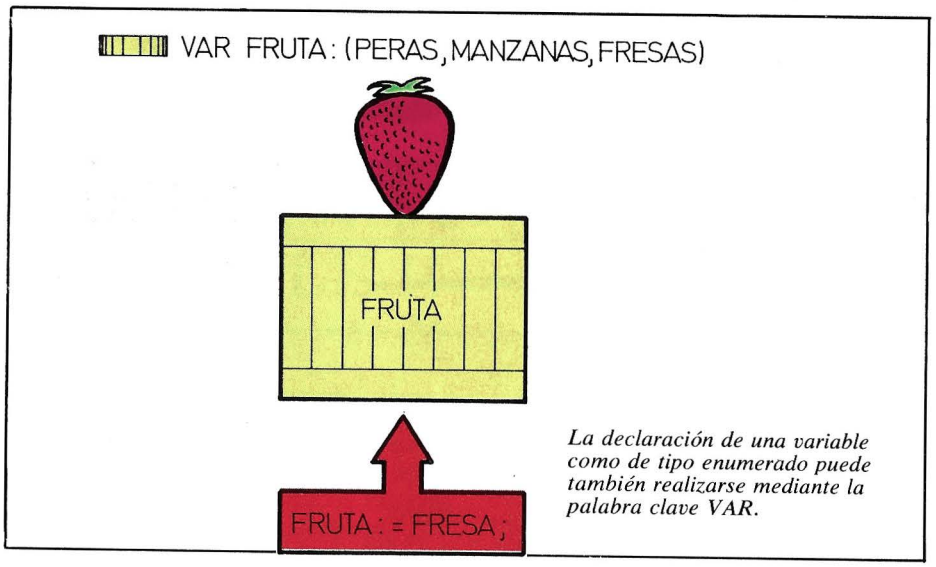
En él, la sentencia de asignación "f:= FRESAS" es válida y plenamente correcta, puesto que la variable "f" ha sido declarada del tipo FRUTAS, entre cuyos posibles valores se encuentra el de FRESAS. Sin embargo, no será posible asignar a "f", o a cualquier otra variable declarada con el tipo FRUTAS, otro valor distinto de los nombres PERAS, MANZANAS o FRESAS; en caso contrario se produciría un error.
La definición de los tipos por enumeración se puede efectuar también directamente de la siguiente forma:

VAR f:(PERAS,MANZANAS,FRESAS);

Un tipo de datos definido por enumeración y que ya está implementado en el lenguaje PASCAL, es el tipo BOOLEAN. Las constantes de este tipo de datos son TRUE y FALSE. Virtualmente, es como si el programador hubiese realizado la siguiente definición:

TYPE BOOLEAN=(TRUE,FALSE)

No obstante, y como quiera que están ya disponibles directamente, el programador no se verá obligado a repetir dicha definición cada vez que desee utilizar el tipo BOOLEAN.



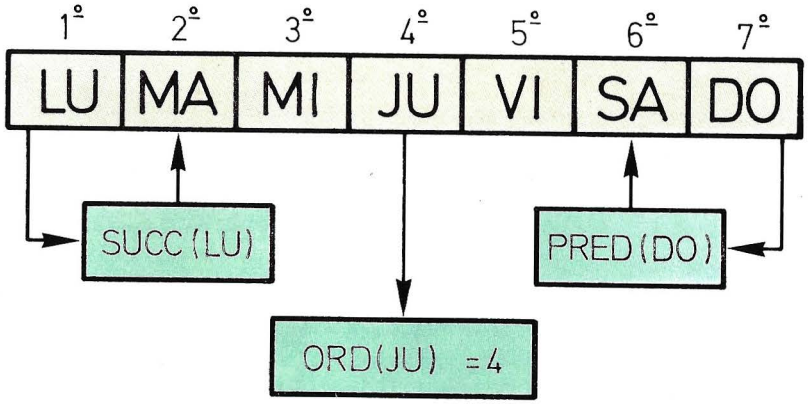
La declaración de una variable como de tipo enumerado puede también realizarse mediante la palabra clave VAR.

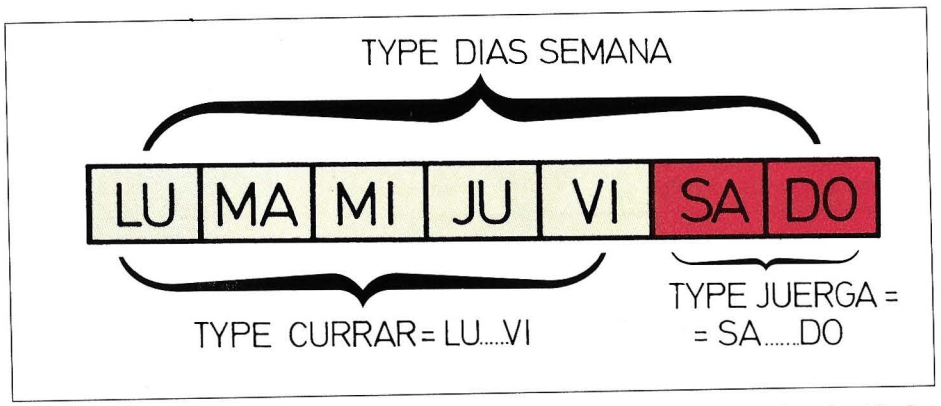
Funciones estándar de los tipos ordinales

Todos los tipos de datos analizados hasta ahora pertenecen a la categoría general de "tipos ordinales". Ello se debe a que, para cualquier valor dado, existe siempre un sucesor y un predecesor del mismo, salvo que se trate del primer o último elemento de alguno de estos tipos.
El lenguaje PASCAL dispone de dos funciones estándar que permiten obtener los valores anterior o posterior a un dato: PRED(X) y SUCC(X); en donde X es un valor de cualquier tipo ordinal.
Supóngase, por ejemplo, que se ha definido el tipo enumerado siguiente:

```
TYPE
      DIASSEMANA=(LUN,MAR,MIE,
        JUE,VIE,SAB,DOM);
```

La función SUCC(LUN) dará como resultado MAR, mientras que la función PRED(SAB) entregará el valor VIE.
Otra función definida para los tipos ordinales es ORD(X). Su objetivo es proporcionar el número entero que indica el orden que ocupa el valor X dentro del conjunto de posibles valores del tipo ordinal al que pertenece.
Otra función estándar, sólo aplicable a datos del tipo CHAR, es CHR(I). Esta devuelve el carácter cuyo número ordinal coincide con el valor indicado por el parámetro de entrada I. Así, la expresión INICIAL:=CHR(ORD('A')+I−1), asignará a la variable INICIAL la letra I−esima del abecedario.
Todas estas funciones permiten proyectar los valores de cualquier tipo ordinal sobre un conjunto de los números naturales y actúan, en consecuencia, como virtuales funciones de transferencia.





Otra forma con la que cuenta el usuario para definir tipos de datos, consiste en la selección de un subconjunto de valores de cualquier otro tipo escalar.

Los identificadores empleados como constantes de un nuevo tipo enumerativo de datos recién definido, no pueden volver a utilizarse como constantes en la definición de otro nuevo tipo de datos. En consecuencia, no será válida la siguiente definición de tipos, pues el tipo al que pertenece el valor SAB queda definido de forma ambigua, al no ser de un tipo único:

```
TYPE
  CURRAR: (LUN,MAR,MIE,JUE,VIE,SAB);
  DESCANSAR: (SAB,DOM);
```

Los tipos definidos por el usuario serán de uso exclusivamente interno del programa. Desde luego, pueden ser asignados o probados; si bien, no se podrán utilizar en sentencias de lectura o escritura, es decir, no se podrán asignar mediante sentencias READ o READLN, ni escribir sus valores con WRITE o WRITELN.

```
PROGRAM CONCIENCIA (INPUT);
TYPE
      DIASSEMANA=(LUN,MAR,MIE,
        JUE,VIE,SAB,DOM);
      LABORABLES=LUN..VIE;
      CANTIDAD=1..5;
      LETRA='A'..'Z';
VAR   HOY:DIASSEMANA;
      COPAS:CANTIDAD;
      CURRAR:LABORABLES;
      NUM:INTEGER;
      INICIAL:LETRA;
BEGIN
      READLN (INICIAL);
      CASE INICIAL OF
      'L':HOY:=LUN;
      'M':HOY:=MAR;
      'X':HOY:=MIE;
      'J':HOY:=JUE;
      'V':HOY:=VIE;
      'S':HOY:=SAB;
      'D':HOY:=DOM
      END;
      CURRAR:=LUN;
      REPEAT BEGIN
        IF HOY=CURRAR THEN
          COPAS:=1
        ELSE COPAS:=5;
        CURRAR:=SUCC(CURRAR)
      END
      UNTIL CURRAR:=VIE OR HOY=
        CURRAR;
      WRITELN('NO BEBER MAS DE
        ',COPAS,' COPAS')
END.
```

TIPOS SUBRANGO

Para definir sus propios tipos de datos, el programador cuenta con un segundo método: la definición de los llamados tipos subrango. Este nuevo tipo de datos, que puede ser creado a partir de cualquier otro tipo de dato escalar definido con anterioridad (excepto del tipo REAL), consiste en un subconjunto de valores de ese otro tipo, limitado por un valor máximo y un valor mínimo. Su definición adopta el siguiente aspecto:

TYPE <identificador tipo>=<cte.>..
<cte.>;

Las dos constantes que delimitan el rango de valores permisible deben ser del mismo tipo, y además deben estar ordenadas: límite inferior a la derecha del límite superior.
Por ejemplo, si se desea utilizar una variable para manipular los números de los días del mes, ésta se puede declarar como de tipo INTEGER. Pero, si por cualquier motivo se asigna a esta variable un valor entero inferior a "1" o superior a "31", obviamente se estará cometiendo un error de concepto que no será detectado por el programa; desde luego, a no ser que se codifique alguna sentencia que se encargue de comprobar tal situación. Esta sentencia no será necesaria si se define a la variable como de un tipo subrango de los números enteros, de la siguiente forma:

```
TYPE DIASDELMES: 1..31;
VAR DIA:DIASDELMES
```

Ahora, la variable DIA sólo podrá tomar los valores de los números enteros comprendidos entre 1 y 31, ambos inclusive. Si se realizara una asignación fuera de ese rango —por ejemplo, DIAS:=40— se produciría un error. Como ya se mencionó, este tipo de datos se puede definir a partir de cualquier otro tipo escalar, excepto del tipo REAL. Por lo tanto, serán válidas las definiciones de tipos que se realizan en el programa adjunto (CONCIENCIA).

Apple DOS 3.3

Introducción al sistema operativo estándar de la familia APPLE II

Después de las afamadas manzanas de Eva, Guillermo Tell, Isaac Newton y la "gran manzana" (denominación dada por los newyorquinos a su ciudad), quizás la manzana más famosa haya sido la que representa a la firma norteamericana APPLE, responsable de la popularización de los ordenadores personales.

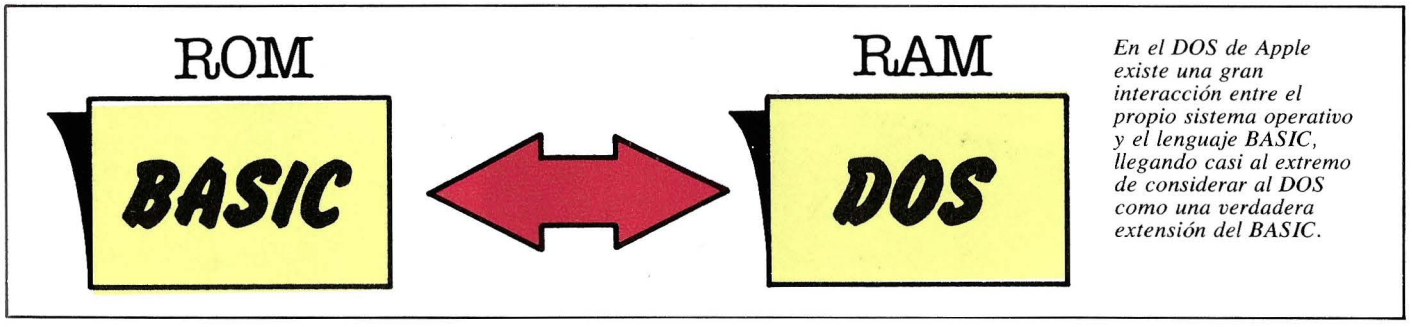
GENERALIDADES

El sistema operativo DOS de Apple nació en 1978 a raíz de la introducción de los disquetes en los modelos de microordenadores que integraban la familia Apple II. En aquellos principios, su diseño se orientaba básicamente hacia discos flexibles de una sola cara y simple densidad.

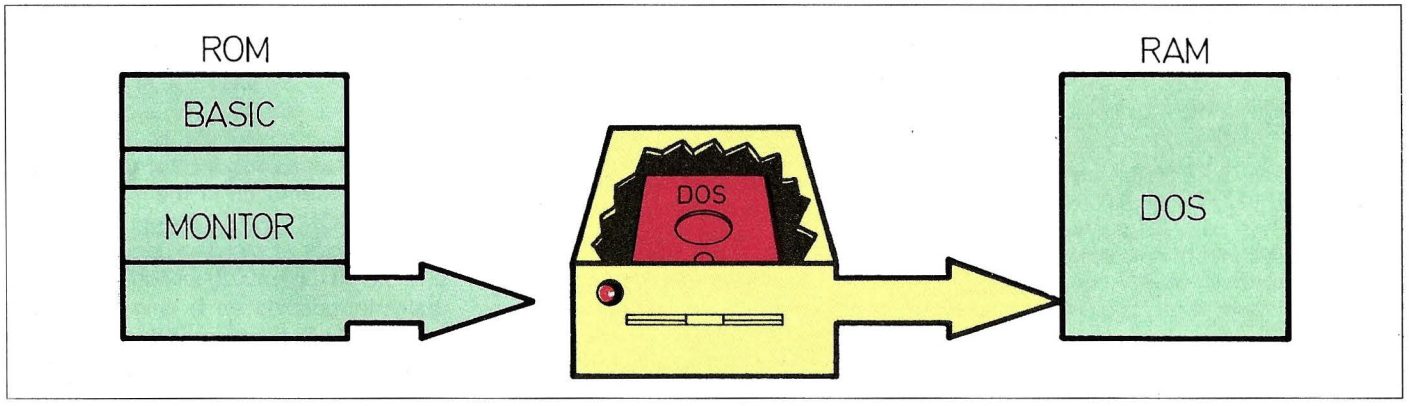
El sistema operativo Apple.DOS está dirigido claramente hacia el campo doméstico, en lugar de hacia el ámbito científico como era el caso de sus predecesores e incluso de sus contemporáneos. En consecuencia, resultaba un sistema operativo simple, con un mínimo de utilidades, lo cual facilitaba su uso y aprendizaje, aunque, por contra, se veía mermada su potencia y variedad de comandos.

La principal peculiaridad del Apple.DOS reside en su fuerte interrelación con el traductor de lenguaje BASIC que equipa al microordenador. Hasta el punto de que éste se ocupa de funciones que normalmente son propias del sistema operativo. Tal es el caso de la propia inicialización o formateo de los disquetes: para ello es necesario cargar en memoria un programa BASIC que realizará esta tarea. Otro ejemplo revelador lo constituye la comunicación con los distintos dispositivos conectados al sistema, tarea que también corre a cargo del lenguaje BASIC mediante los comandos PR# e IN#.

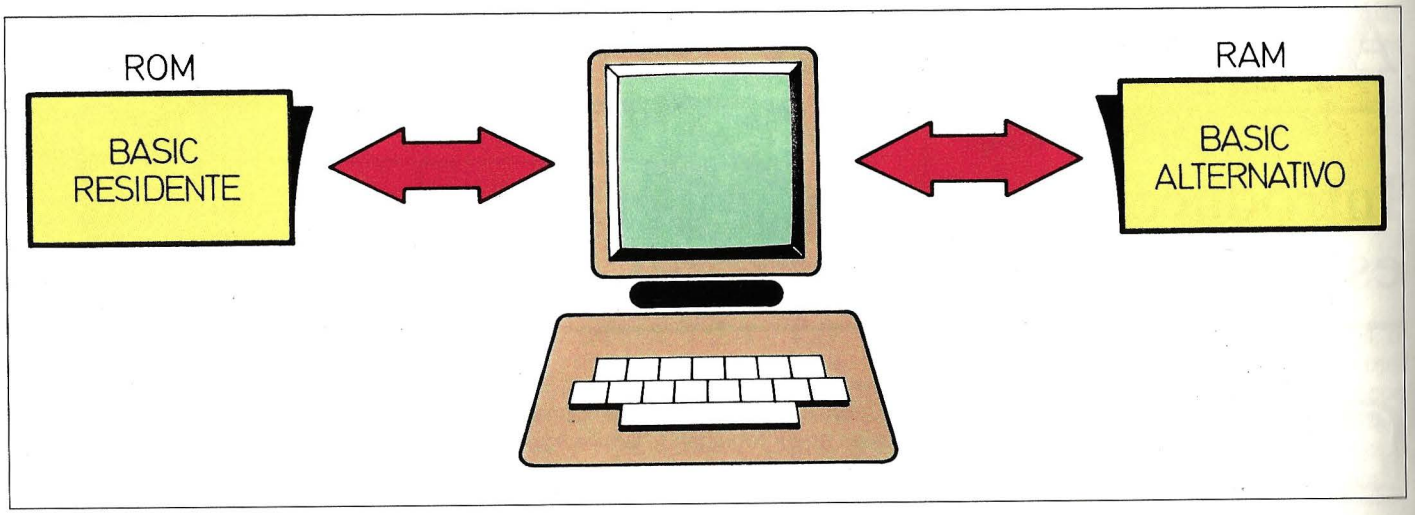
El Apple DOS 3.3 no es un sistema operativo especialmente optimizado por lo que respecta al manejo de los discos. En efecto, su acceso al disco no es muy rá-



En el DOS de Apple existe una gran interacción entre el propio sistema operativo y el lenguaje BASIC, llegando casi al extremo de considerar al DOS como una verdadera extensión del BASIC.



El Monitor verifica la existencia de una unidad de disco asociada al equipo y, tras ello, carga en la memoria central del ordenador el sistema operativo residente en el disco.



Dependiendo del modelo de ordenador, es posible contar con dos versiones de lenguaje BASIC en ROM: Integer y Applesoft. El sistema puede disponer también de un traductor de BASIC en RAM si la extensión de dicha memoria lo permite.

Para que el sistema operativo DOS 3.3 pueda tomar el control de un ordenador de la serie Apple II (o compatible), es preciso que éste cuente con un mínimo de 16 Kbytes de memoria RAM, e incorpore una unidad de disco.

pido, y también es escasa su flexibilidad para el tratamiento de los posibles discos rígidos que puedan conectarse al sistema. Estos últimos no pueden ser gestionados si previamente no se subdividen en disquetes "lógicos". Por lo demás, estos disquetes lógicos son incapaces de contener a un fichero que exceda su capacidad, limitando de este modo la longitud de los ficheros en el sistema. A ello, cabe añadir que esta excesiva compartimentación del espacio en discos conlleva un apreciable desperdicio de la zona utilizable para almacenamiento.

REQUERIMIENTOS HARDWARE

El sistema operativo DOS 3.3 de Apple necesita un soporte material sobre el cual ejercer sus funciones, aunque bien es cierto que sus necesidades son mínimas. El uso de este sistema operativo exige que el microprocesador sobre el que va a operar sea un Apple II, Apple II plus, Apple IIe, o compatibles, basados todos ellos en el microprocesador de 8 bits 6502. La zona de memoria de acceso aleatorio (RAM) debe ser, al menos, de 16 Kbytes. Y, por último, es necesaria la presencia de una unidad de disco. Esta hará posible la carga del sistema operativo y permitirá al mismo realizar las tareas que le son propias.

Todos estos son requisitos mínimos para empezar a funcionar. Si bien, es aconsejable disponer de una mayor capacidad de memoria RAM, puesto que al estar reservados 10,5 Kbytes para el uso del sistema operativo, el usuario conservará tan sólo 5,5 K para sus programas. Asimismo, la presencia de más de una unidad de disco permitirá un manejo más cómodo de los ficheros, al no tener que estar intercambiando constantemente los disquetes con los que se trabaja.

ARRANCANDO EL DOS

Al conectar el microordenador a la red, se pone en marcha automáticamente un programa residente en la memoria de sólo lectura (ROM). Este programa, denominado Monitor pasará a cargar en memoria el sistema operativo, leyéndolo del disquete introducido en la unidad de disco número 1. Esta operación se llevará a cabo con éxito siempre y cuando el disquete en cuestión contenga el sistema operativo.

Una vez cargado el Apple.DOS en memo-

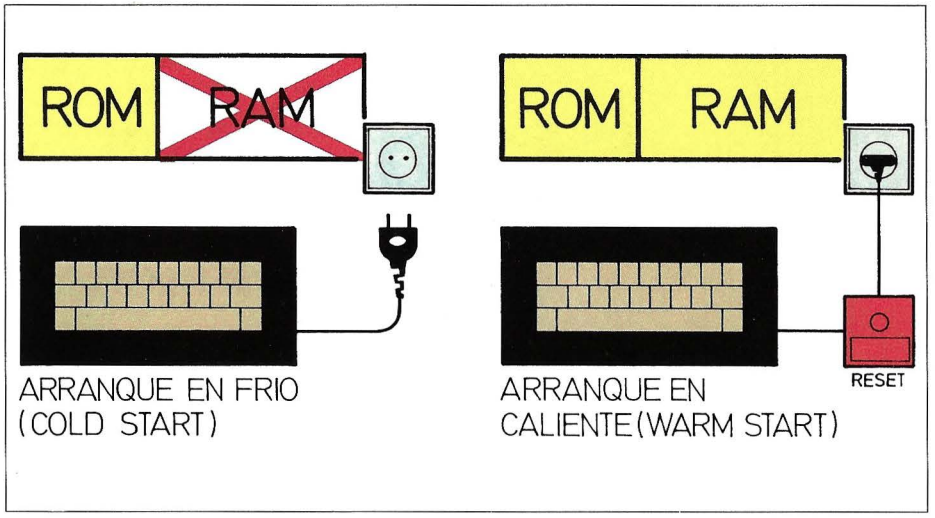
ria, éste es activado por el Monitor. En ese instante, el DOS asumirá el control de las comunicaciones del sistema con su entorno, función asignada hasta el momento al programa Monitor.

BASIC APPLESOFT E INTEGER

El lenguaje BASIC disponible no es único, sino que depende de la capacidad de cada modelo de microordenador así como del hardware y firmware (programas almacenados permanentemente en ROM) que lo equipen. Dos son los traductores de lenguaje BASIC primordiales: uno está almacenado en la memoria ROM del sistema, el llamado BASIC residente, mientras que el segundo de ellos se encuentra grabado en un disquete, el BASIC alternativo, variando de unos equipos a otros. Así pues, para Apple II el BASIC residente es el BASIC Integer, mientras que para los modelos Apple II plus y Apple IIe el BASIC almacenado en ROM es el BASIC Applesoft.
El sistema operativo verifica si el espacio de memoria RAM disponible es suficiente como para contener un BASIC alternativo; y en el caso de que sea posible, busca en el disquete dicho programa. A continuación, el DOS ejecuta un programa (HELLO) cuya misión es la de mostrar la versión del sistema operativo en uso, y cargar el BASIC alternativo desde el disquete, en el supuesto de que dicha operación pueda llevarse a cabo. Como último paso, se imprime en la pantalla el "prompt" o indicador de presencia del sistema. Este coincidirá con el símbolo] si es el BASIC alternativo el que está activo, o con el símbolo > en el caso de que lo esté el BASIC residente. Uno u otro símbolo estará seguido por el cursor. A partir de este instante, los distintos comandos del sistema operativo, así como los propios del lenguaje BASIC activado, quedan a disposición del usuario para su uso inmediato.

ARRANQUES FRIOS Y CALIENTES

En ciertos casos puede ocurrir que debido al mal funcionamiento de un programa o



En el caso de que el sistema operativo quedara bloqueado, el DOS de Apple autoriza dos procedimientos para su reinicialización: "arranque en frío" con el que se pierde irremisiblemente la información almacenada en la memoria RAM; o "arranque en caliente", método que preserva la integridad de la información memorizada en RAM.

Comunicación periférica

La comunicación de un microordenador de la serie Apple II con los diversos periféricos, tales como monitor, unidad de disco, teclado, impresora, trazador gráfico, modem, etc., se establece a través de la conexión de una tarjeta que aporta la circuitería necesaria para el control del dispositivo periférico. Estas tarjetas se insertan en los contenedores o "slots" presentes en los microordenadores Apple II, Apple II plus o Apple IIe.
En todo caso, la inclusión de estos controladores no resuelve el problema en su integridad; falta aún el software que permita la comunicación con el exterior. Software que facilite un diálogo semejante al que permiten las sentencias PRINT e INPUT del lenguaje BASIC, cuya actuación hace posible mandar información al monitor o pantalla y recibirla desde el teclado.
En el DOS 3.3 nos serviremos de sentencias similares a las anteriores para lograr la comunicación con otros dispositivos; éstas con PR# e IN#.
El comando PR# indica al ordenador cuál es el dispositivo destinatario de toda la información que salga del mismo (listados, mensajes de error, comandos, peticiones o prompts, etc.), hasta que se especifique un nuevo dispositivo con el uso del mismo comando. La sintaxis del comando es:

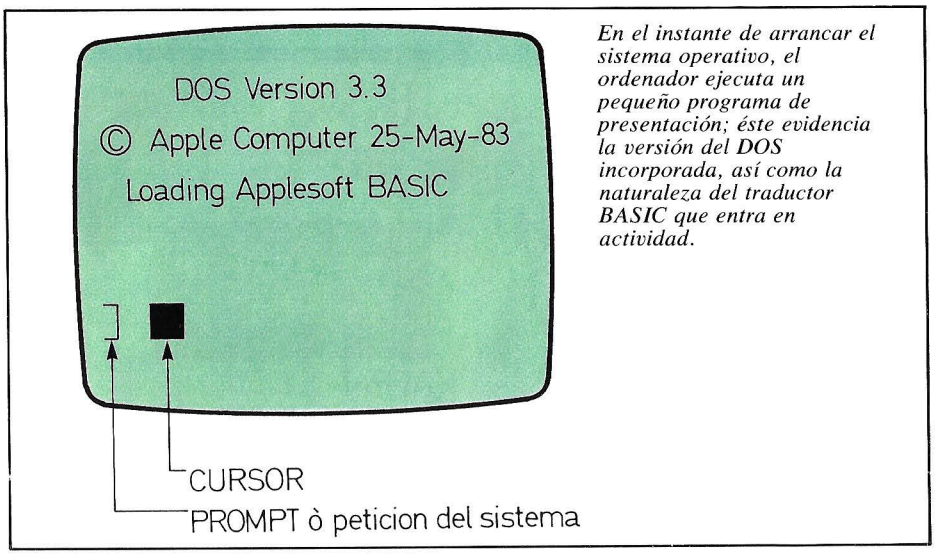
PR# n (con $0 \leq n \leq 7$)

El número n hace referencia al conector o "slot" al cual está conectada la tarjeta que controla al dispositivo.

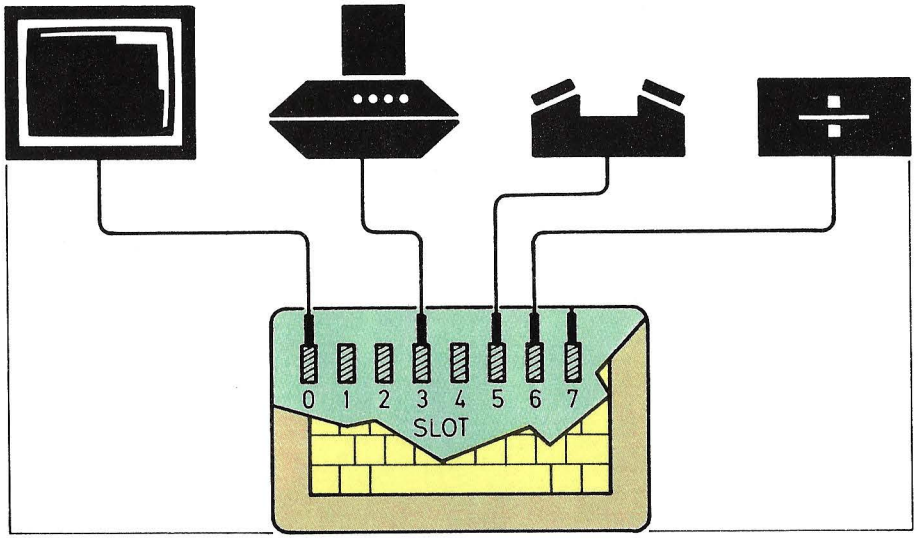
La recogida de información se realiza con el comando IN#. Este pone en conocimiento del ordenador cuál es el dispositivo periférico que va a mandarle información. Su sintaxis es idéntica a la del comando anterior, utilizándose un número del 0 al 7 para indicar el dispositivo conectado al "slot" de expansión. Por ejemplo:

IN# 4

En general, cualquier conector puede acoger a una tarjeta de control destinada a cualquier dispositivo. No hay un número que señale inequívocamente un tipo de unidad de control, excepto el cero que se reserva para el teclado (IN# 0) y para la pantalla (PR# 0), o el 3 que suele utilizarse como indicativo de la impresora.
Al respecto, hay una excepción que merece la pena citar. Se trata de la situación producida al utilizar en un Apple IIe tarjetas que permiten el uso de un juego de caracteres apropiado para la visualización de 80 caracteres por línea. De utilizar el comando PR# 0 para volver al juego de caracteres normal, es muy probable que el sistema operativo se venga abajo; en tal caso es recomendable el uso de PR#3 para direccionar de nuevo la salidad de información hacia la pantalla.
Si en un comando PR# o IN# se hace referencia a un número de conector que no contenga ninguna tarjeta de control externo, el sistema operativo DOS quedará bloqueado, y será necesario arrancarlo de nuevo para poder continuar la sesión de trabajo.



En el instante de arrancar el sistema operativo, el ordenador ejecuta un pequeño programa de presentación; éste evidencia la versión del DOS incorporada, así como la naturaleza del traductor BASIC que entra en actividad.



Los dispositivos periféricos son gestionados a través de las respectivas tarjetas de control, conectadas a los "slots" dispuestos en el cuerpo del ordenador.

de un dispositivo periférico, no sea posible para el usuario salir del atolladero en el que está metido y regresar al DOS. Tal situación hace necesaria una restauración del sistema operativo. El método recomendado es el arranque en caliente o "warm start", lo que se consigue al accionar simultáneamente las teclas <CONTROL> y<RESET>. El resultado es una nueva inicialización del sistema operativo, pero con la ventaja de que el programa almacenado en memoria no se pierde; concretamente, su efecto equivale a ejecutar la sentencia BASIC CALL 1002.
Por el contrario, si se teclea PR#6 para restaurar el sistema operativo con un arranque en frío ("cold start"), o si se desconecta el microordenador de la fuente de alimentación, el contenido de la memoria RAM se pierde irremediablemente.

CONECTORES, UNIDADES DE DISCO Y VOLUMENES

La localización de los distintos dispositivos y ficheros se realiza, bajo el control del Apple.DOS, referenciándolos respecto a conectores de expansión (slots), unidades de disco (drives) y volúmenes, con objeto de identificarlos unívocamente.
Los "slots" son conectores internos que permiten la incorporación de tarjetas de circuito impreso, cuyo principal cometido es habilitar la comunicación del ordenador con dispositivos periféricos (unidades de disco, impresoras, ...).
Para hacer referencia a un dispositivo externo específico, es necesario indicar el "slot" al que está conectada la tarjeta de expansión que lo controla (del número 0 al 7); a no ser que su emplazamiento se haya especificado con anterioridad o que el "slot" en cuestión sea el que toma por defecto el sistema operativo.
El volumen hace referencia a la etiqueta con la que ha sido marcado el disquete en el proceso de inicialización. Esta puede coincidir con un número comprendido entre 1 y 254; éste último es el que se toma por defecto, a no ser que se especifique un número de volumen diferente.
Al hacer referencia a un fichero, hay que tener en cuenta que el volumen especificado ha de estar en consonancia con el que ostenta el disquete que lo contiene; en tal situación, se toma por defecto el volumen 254 o el coincidente con el número del último volumen citado.
Empleando estos tres identificadores se logra definir, sin ambigüedad alguna a todos los dispositivos conectados al sistema, así como a los ficheros contenidos en los disquetes. Como ejemplo cabe citar los siguientes:

S6, D1: slot 6 drive 1
S6, D2: slot 6 drive 2
S5, D1: slot 5 drive 1
S6, D1, V20: slot 6 drive 1 volumen 20

Para conocer en todo momento cuáles son los números de conector y de unidad de disco actualmente en uso, existe la utilidad SLOT# capaz de mostrar dichos valores. Ello evitará posibles errores que pueden ser cometidos al emplear referencias erróneas en una operación. El mencionado programa reside en el disquete que contiene el sistema operativo, etiquetado normalmente como SYSTEM MASTER DISK, junto con otras utilidades. Su ejecución se activa mediante el comando RUN, al igual que cualquier otro programa. Por ejemplo:

```
] RUN SLOT#
  CURRENT SLOT IS 6 AND DRIVE IS 1
]
```

Manteniendo el criterio adoptado en capítulos anteriores, se considerará que el texto en letra cursiva es el que introduce el usuario.

Multiplan (1)

La potencia de una hoja electrónica

En este capítulo da comienzo el estudio de una nueva hoja electrónica; se trata del programa MULTIPLAN, creado por la firma americana MICROSOFT. De él existen múltiples versiones, destinadas a distintos sistemas operativos y equipos entre los que cabe destacar el CP/M, MS/DOS, APPLE/DOS y APPLE MACINTOSH. MULTIPLAN es, sin duda, uno de los paquetes de gestión más extendidos en todo el mundo, debido a que sintetiza una considerable potencia de cálculo con una notable sencillez de uso.

INTRODUCCION AL MULTIPLAN

MULTIPLAN se puede definir como un programa destinado a resolver prácticamente cualquier tipo de cálculo. Al igual que en las restantes hojas electrónicas, su base de trabajo es una matriz constituida por un determinado número de celdas; en este caso, formado por 255 líneas y 63 columnas.

El usuario se relaciona con el programa a través de la representación en la pantalla del ordenador de lo que cabe denominar "la hoja de trabajo". En ella se pueden introducir datos numéricos o literales; insertar o eliminar espacios entre los datos; reservar posiciones en el "papel de trabajo" para que en ellas el programa calcule automáticamente su valor, etc. En resumen, el programa MULTIPLAN participa de todas las características generales de las hojas electrónicas.

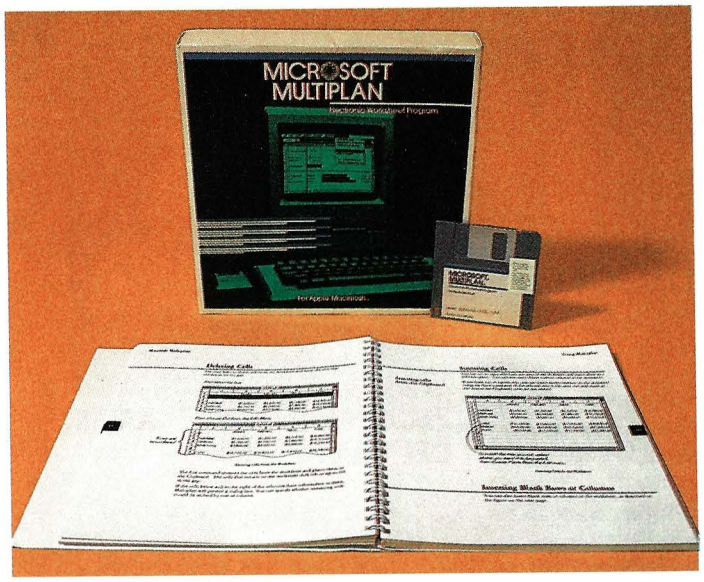
Para facilitar la labor del usuario, el programa brinda un sistema de ayudas (HELP) que puede ser invocado en cualquier instante. Ayudas que ofrecen al operador explicaciones interactivas adecuadas para resolver sus dudas puntuales.

En un párrafo anterior se apuntaba que el usuario trabaja sobre una virtual "hoja de trabajo", una de cuyas zonas aparece representada en la pantalla. Evidentemente, sobre este "papel" resulta imposible escribir físicamente con un lapicero; por lo tanto, la información se debe introducir a través del teclado. Esta información puede coincidir con datos (números o palabras alfanuméricas) o bien con funciones de cálculo que permitirán al programa calcular y recalcular los valores que puedan sintetizarse a partir de otros datos presentes en la hoja de trabajo.

PECULIARIDADES TECNICAS DEL MULTIPLAN

La diferencia más notoria entre el MULTIPLAN y otras hojas electrónicas se advierte en la forma en la que se numeran los elementos de la matriz. El método más extendido consiste en identificar las columnas mediante letras y las filas mediante números; con ello, cada celda queda perfectamente localizada por un par de valores: una letra y un número. En

La hoja electrónica MULTIPLAN es una poderosa herramienta para cualquier usuario que tenga la necesidad de realizar con frecuencia múltiples cálculos. De ahí que su presencia sea extensible tanto al ámbito del profesional independiente como al de la empresa.



La aplicación MULTIPLAN es una hoja electrónica creada por la firma americana Microsoft. Junto con el paquete VISICALC, detenta el liderazgo en el terreno de las aplicaciones de hoja electrónica para microordenadores.

La hoja electrónica MULTIPLAN está disponible en distintas versiones, adaptadas a los sistemas operativos y ordenadores más populares.

cambio, MULTIPLAN se adapta a la notación matemática, es decir, numera los elementos de la matriz mediante parejas de números: el primero indica la fila y el segundo la columna. De esta forma, el elemento de la hoja electrónica situado en la intersección de la 5.ª fila y la 3.ª columna, será identificado como (5, 3).

No existe ningún ordenador personal que disponga de una pantalla con dimensiones suficientes como para visualizar simultáneamente las 255 líneas y 63 columnas que forman la hoja electrónica. La solución que aporta el programa MULTIPLAN es la tradicional: la pantalla juega el papel de una "ventana" que puede desplazarse sobre la hoja electrónica según las necesidades del usuario.

El aspecto físico de la pantalla de trabajo del MULTIPLAN está integrado por cuatro elementos fundamentales.

1—En la zona superior aparecen los números correspondientes a las columnas que están presentes en la "ventana" actualmente visible.

2—En la parte lateral izquierda, figuran los números de las filas que están presentes en la "ventana" activa.

3—Todo el espacio comprendido entre los números identificadores de filas y columnas, está ocupado por los elementos de la matriz (también denominados celdas).

4—En determinada zona de la pantalla (en el borde superior o inferior, según la versión del MULTIPLAN), aparecerá un menú de opciones entre las que el usuario podrá elegir; habitualmente, situando sobre ellas el cursor y pulsando la tecla RETURN o ENTER (según el ordenador personal utilizado).

ELEMENTOS DEL MULTIPLAN

Fundamentalmente, cabe distinguir cuatro elementos básicos en el programa MULTIPLAN, elementos que se detallan a continuación:

- *Comandos*

Sirven para gestionar la hoja electrónica y su puesta en práctica exige cuatro pasos sucesivos:

1—Seleccionar una celda de la hoja electrónica; para ello basta con desplazar el cursor, mediante las teclas apropiadas según el ordenador utilizado, hasta el lugar deseado.

2—Seleccionar el comando a ejecutar. Para realizar este paso se puede recurrir, normalmente, a cualquiera de las siguientes alternativas:

— Teclear el nombre del comando deseado (con la primera letra suele ser suficiente).

— Desplazar el cursor a través del menú de comandos que aparece en la parte inferior de la pantalla, seleccionar el comando oportuno, y pulsar la tecla RETURN.

3—Responder a las solicitudes de argumentos que siguen al reconocimiento del comando.

4—Por último, se accionará la tecla RETURN para dar por finalizada la ejecución del comando; o en cualquiera de los tres primeros pasos se pulsarán las teclas CTRL y C, simultáneamente, para cancelar su ejecución.

- *Funciones*

Sirven para construir fórmulas cuyo efecto será calcular y/o recalcular los valores de las celdas variables. Para definir las funciones se pueden utilizar cualquiera de los siguientes conceptos:

1—Números; fundamentalmente empleados para cálculos numéricos, pueden ser enteros o decimales (por ejemplo: 12 ó 17.39).

2—Literales; constituidos por cadenas de caracteres que en algunos casos también pueden incluir números. Por ejemplo, para indicar una fecha se podría utilizar el literal: "10-02-1955". Nótese que los literales se encierran entre comillas (").

3—Referencias de celda, mediante el uso de expresiones del tipo: RnCm; en donde la letra R (ROW) significa línea, la letra C (COLUMN) columna, y los números n y m señalan la posición de la fila y de la columna, respectivamente. Por ejemplo,

mediante R3C9 identificamos a la celda situada en la tercera fila y novena columna.

4—Valores lógicos, que como ya hemos visto en otras ocasiones, pueden tomar únicamente los valores verdadero (TRUE) o falso (FALSE).

5—Valores de error; éstos sirven para identificar un error cometido en la introducción o cálculo de una celda. Por ejemplo, si especificamos como fórmula una división en la que el divisor sea cero, la evaluación resultará imposible de realizar.

Por último, es imprescindible utilizar una función determinada que ligue a los anteriores conceptos.

- *Hoja o papel de trabajo*

Ya descrita anteriormente, sirve para que el usuario pueda representar, en forma de matriz, los datos que manejará mediante el programa.

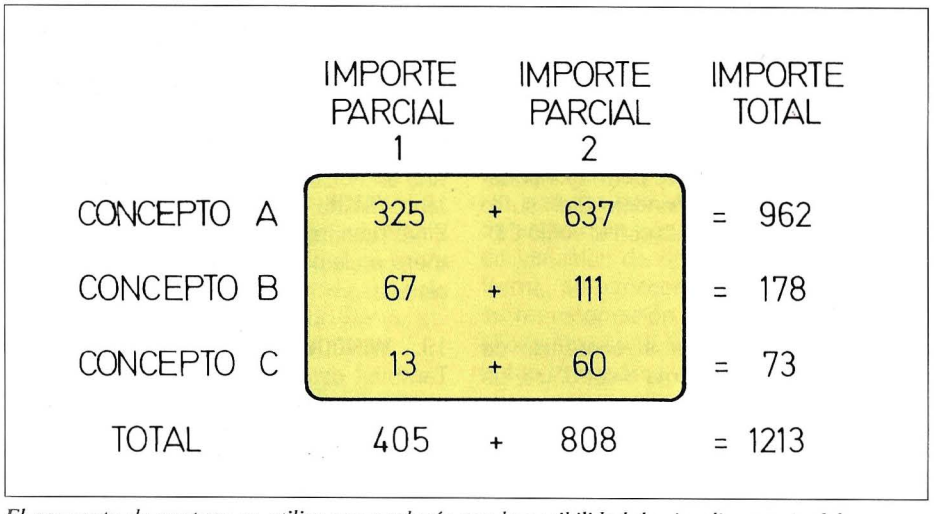
- *Ficheros*

Después de haber trabajado con la hoja electrónica, el usuario puede desear almacenar su contenido; bien para su uso posterior con los mismos y otros datos, o bien para modificar las fórmulas y, por lo tanto, el esquema de funcionamiento de la hoja electrónica. Para ello existen dos operaciones básicas:

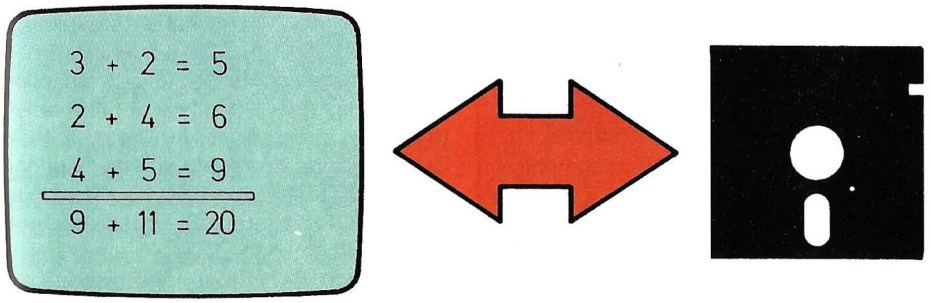
1—Lectura de una hoja electrónica almacenada en un disco flexible o rígido y carga en la memoria principal, de forma que el usuario pueda comenzar a utilizarla inmediatamente.

2—Escritura de una hoja electrónica, residente en la memoria principal, sobre un soporte de almacenamiento externo, ya sea disquete o disco rígido, para su posterior utilización.

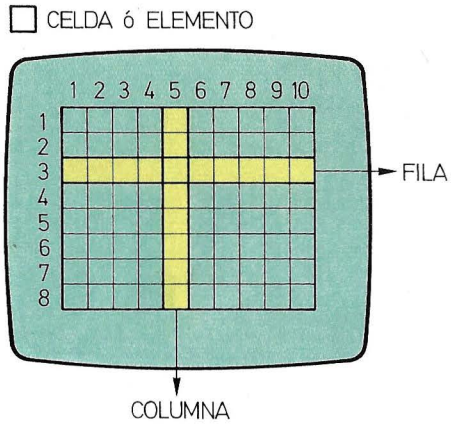
Si bien los cuatro elementos citados anteriormente constituyen la base de cualquier hoja electrónica, los dos primeros (comandos y funciones) son los que permiten diferenciar unas de otras. Por lo tanto, nuestro estudio se volcará, especialmente, en la descripción de los comandos y funciones del MULTIPLAN.



El concepto de ventana se utiliza por analogía con la posibilidad de visualizar parte del contenido de un recinto (hoja electrónica, en este caso). La diferencia respecto a su acepción más literal, reside en que la "ventana" de la hoja electrónica puede moverse libremente sobre la superficie total.



Como quiera que MULTIPLAN es una aplicación nacida en el entorno microinformático, su repertorio de comandos incluye órdenes para el almacenamiento y recuperación de hojas electrónicas en disco magnético.



Los elementos o células de la hoja electrónica se identifican por los números que definen a la fila y columna en cuya coincidencia se encuentran.

RESUMEN DE LOS COMANDOS DEL MULTIPLAN

Dentro del programa MULTIPLAN existen veinte comandos distintos (recientes versiones pueden incluir algún comando adicional). Algunos de ellos ofrecen varias posibilidades que se especifican mediante un sufijo. Con objeto de no extendernos demasiado en la exposición de los comandos, éstos se esquematizarán brevemente. Para su estudio riguroso se puede consultar el "Manual de referencia" que junto con la "Guía del Usuario", forma la documentación que se adjunta con el disco que almacena la aplicación.

1. **ALPHA**

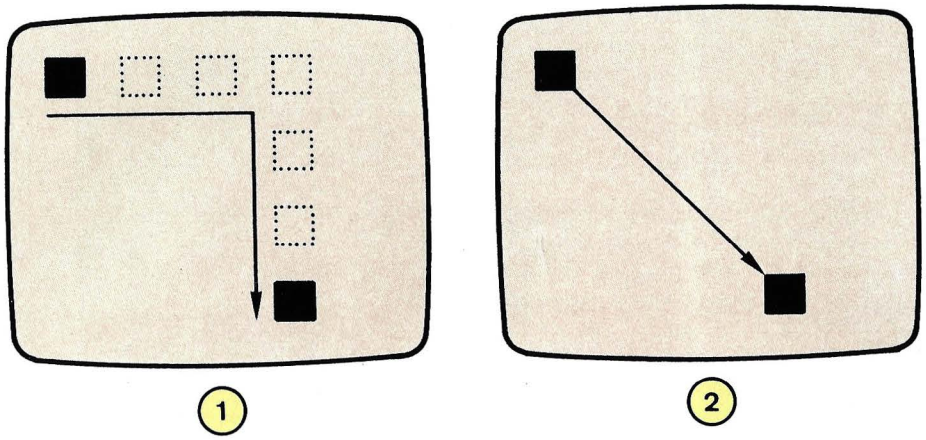
Sirve para especificar que el contenido de un elemento concreto de la hoja electrónica será de tipo alfabético, y permitirá introducir el texto sin comillas.

2. **BLANK**

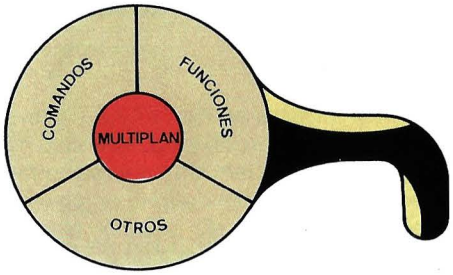
Sirve para reemplazar el contenido de la celda activa por espacios en blanco; también puede utilizarse para borrar el contenido de un grupo de celdas de la hoja electrónica.

3. **COPY**

Tiene tres variantes que se ejecutan mediante las órdenes COPY RIGHT, COPY DOWN y COPY FROM. Sirven para reali-



Para emplazarse a través de la hoja electrónica, el usuario cuenta con dos procedimientos: recorrer un camino —elemento a elemento— desde el punto de origen hasta el de destino, haciendo uso de las teclas para el desplazamiento del cursor, o "saltar" directamente desde la celda de origen a la de destino por medio del comando GOTO.



Dentro de las herramientas de trabajo que brinda la aplicación MULTIPLAN, los dos grupos más relevantes los constituyen los comandos y las funciones.

zar copias del contenido de un elemento o conjunto de elementos de la hoja en otras posiciones distintas de las originales.

4. **DELETE**
Borra el contenido de una fila o de una columna de la hoja electrónica. Para decidir si se trata de una fila o de una columna, se incluirán los sufijos ROW o COLUMN, respectivamente.

5. **EDIT**
Se utiliza para editar la fórmula que permitió obtener los datos calculados de la celda activa.

6. **FORMAT**
El comando FORMAT, seguido por cualquiera de los sufijos que admite, se emplea para indicar el tipo de letra o dígitos con que deben representarse los datos.

7. **GOTO**
Adecuado para situar el cursor en la celda deseada, sin necesidad de ordenar el desplazamiento paso a paso mediante las teclas de movimiento del cursor.

8. **HELP**
Mediante este comando el usuario puede solicitar información de ayuda al programa.

9. **INSERT**
Su utilidad estriba en que permite insertar nuevas filas acompañándolo del sufijo ROW y nuevas columnas con el sufijo COLUMN.

10. **LOCK**
Su misión es proteger el contenido de algunas celdas de la hoja electrónica, de forma que no pueda ser alterada o borrada por accidente.

11. **MOVE**
Tiene dos posibles sufijos: ROW y COLUMN. Se utiliza para mover —esto es: eliminar en origen y situar en destino— una fila o columna, respectivamente.

12. **NAME**
Este comando sirve para asignar un nombre a un elemento o conjunto de elementos de la hoja electrónica, de tal forma que se facilite su manejo mediante otros comandos o fórmulas.

13. **OPTIONS**
Mediante el sufijo RECALC:YES/NO, se puede ordenar el recálculo automático o manual de las celdas evaluables mediante fórmulas; mediante el sufijo MUTE:YES/NO, se puede activar o desactivar la alarma audible.

14. **PRINT**
Tiene diversos sufijos que permiten, en general, realizar impresiones de la hoja electrónica en varios dispositivos de salida (impresoras o unidades de almacenamiento).

15. **QUIT**
Invocando a este comando finaliza la sesión de trabajo con el programa MULTIPLAN. Como quiera que su activación por error podría resultar nefasta, el programa solicita confirmación.

16. **SORT**
El referido comando sirve para clasificar la información contenida en la hoja electrónica. Para ello es necesario indicar por qué columna se clasificará, desde qué fila hasta qué fila, y si el orden será ascendente o descendente.

17. **TRANSFER**
Dispone de seis sufijos distintos: LOAD, SAVE, CLEAR, DELETE, OPTIONS y REAME, que permiten transferir información desde la memoria principal a dispositivos de almacenamiento, o viceversa.

18. **VALUE**
Sirve para introducir una fórmula o un número en la celda activa de la hoja electrónica.

19. **WINDOW**
También este comando puede ser complementado por diversos sufijos que, en general, permiten definir distintas ventanas que pueden visualizarse simultáneamente en la pantalla del ordenador.

20. **EXTERNAL**
El orden de todos los comandos descritos hasta ahora era estrictamente alfabético. Hemos dejado para el final el denominado EXTERNAL. Ello se debe a que los primeros diecinueve comandos se pueden invocar sin más que especificar la primera letra de su nombre (todas son distintas). Con objeto de no repetir la letra E, el comando EXTERNAL se ejecuta tecleando la letra X, de forma que no se pueda confundir con el comando EDIT.
El comando EXTERNAL, que dispone de varios sufijos, sirve para establecer relaciones entre la hoja electrónica activa y otras hojas de trabajo externas.

Presentación de datos

El comando PRINT USING

En uno de los primeros capítulos de la obra se introdujo el comando PRINT. Este comando ha sido utilizado en la práctica totalidad de los capítulos de este curso de BASIC. Su importancia resulta obvia, pues casi todo proceso exige la presentación de un resultado en pantalla.

Como ya se comentó anteriormente, la presentación de datos debe ser objeto de un especial cuidado. Una presentación que adolezca de estética puede anular incluso la efectividad de un determinado programa. Para evitarlo, cabe echar mano a las facilidades que ofrece el comando PRINT.

Un ejemplo del uso del comando PRINT puede consistir en la visualización de una lista de nombres y números de teléfono. Para ello se partirá de dos matrices N$() y T(), en las que se encuentran los datos. La primera rutina encargada de mostrar en pantalla estos datos puede ser la siguiente:

```
10 FOR I=1 TO 5
20 PRINT N$(I);T(I)
30 NEXT I
■
```

En ella se utiliza un bucle FOR/NEXT para recorrer los elementos de las variables de conjunto. La línea 20 es la encargada de mostrar los datos en pantalla. El separador empleado en el argumento de PRINT es un punto y coma (;), éste hará que los datos se presenten contiguos. Un posible resultado de la ejecución de esta rutina es el que se muestra a continuación:

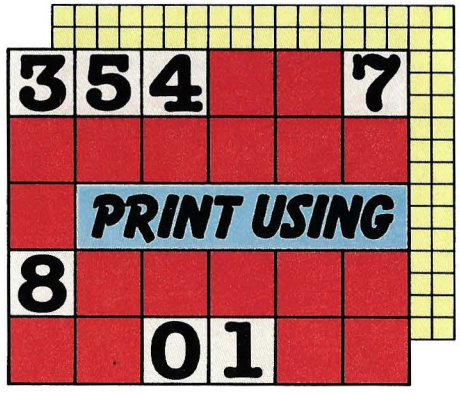
```
RUN
MANOLO 2.22334E+06
LUIS 2.34110E+06
PEDRO4.44223E+06
CARLOS 2.21122E+05
JUAN 4.00554E+06
■
```

Como puede apreciarse, el resultado no es el deseado. Los números de teléfono aparecen en forma exponencial, y lo necesario es que se muestren con todos sus dígitos.

Un método para mejorar la presentación de estos números, pasa por almacenarlos en variables de doble precisión. De esta forma, se almacenarán con más cifras y su representación será la adecuada. Para ello basta con definir T() como variable de doble precisión:

```
5 DEFDBL T
```

Con la adición de esta nueva línea, el resultado de la ejecución tomaría este nuevo aspecto:



El uso de la función USING asociada al comando PRINT permite especificar el formato en el que deben ser presentados los datos.

```
RUN
MANOLO 2223344
LUIS 2341100
PEDRO 4442233
CARLOS 221122
JUAN 4005544
■
```

De todas formas, el formato de salida aún no es el ideal. Cabría desear una presentación en la que los números de teléfono aparecieran en la misma columna de la pantalla. Algo similar a lo siguiente:

```
MANOLO    2223344
LUIS      2341100
PEDRO     4442233
CARLOS     221122
JUAN      4005544
```

A lo largo del presente capítulo se describen varios métodos adecuados para lograr una presentación de este tipo.

METODOS DE TABULACION

El método más directo para separar datos en columnas, lo protagoniza el uso de la coma como separador en el argumento de PRINT. A diferencia con el punto y coma, la coma permite saltar un número fijo de espacios. De esta forma se consigue situar a los datos en diferentes columnas. Utilizando esta posibilidad, la línea 20 quedaría de la siguiente forma.

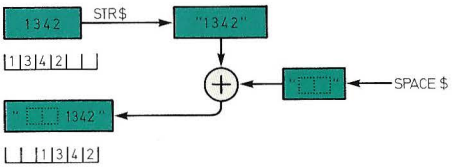
```
20 PRINT N$(I),T(I)
```

Este cambio afectaría a la presentación en pantalla, produciendo un aspecto más regular.
Un mejor resultado se obtiene haciendo uso de la cláusula TAB dentro del argumento de PRINT. Como se recordará, esta función asociada al comando PRINT permite especificar una tabulación. Es decir, el dato que acompaña a TAB indica el número de espacios a partir del margen izquierdo que se han de saltar para imprimir el siguiente dato. Con la intervención de TAB, se puede colocar la segunda columna a la distancia que desee el usuario. Un posible resultado de su actuación puede ser el siguiente:

```
RUN
MANOLO      2223344
LUIS        2341100
PEDRO       4442233
CARLOS      221122
JUAN        4005544
■
```

Tal presentación en pantalla correspondería a la ejecución del programa con la siguiente línea 20:

```
20 PRINT N$(I);TAB (10);T(I)
```



Un posible método para justificar los números a la derecha es convertirlos previamente en cadenas de caracteres y tratarlos a continuación.

Al emplear ambos métodos se descubre un problema. El cuarto dato es un número de teléfono con menos cifras que los restantes. Como se observa, los datos quedan justificados por la izquierda, en algunos casos sería de desear que éstos quedaran justificados por la derecha.
Un ejemplo típico de esta necesidad lo constituyen las listas de precios; en ellas conviene que las unidades queden alineadas. Para solventar este inconveniente se puede hacer uso de las funciones que proporciona el BASIC para el tratamiento de cadenas.

FORMATEAR NUMEROS COMO CADENAS

Un método para justificar un dato numérico a la derecha consiste en convertirlo en un dato de tipo cadena, y rellenar las posiciones a la izquierda con espacios en blanco. Lo primero que hay que hacer es conseguir una cadena cuyos elementos sean las cifras del número dado. Esto es precisamente lo que se obtiene con el uso de la función STR$. Así, pues, la primera línea de programa que contendrá esa función aparecerá como sigue:

```
15 LET T$=STR$(T)
```

La adición de espacios en blanco se efectúa por medio de una operación de concatenación de cadenas (+). Si se crea una cadena con el número adecuado de "blancos" y se almacena en B$, la línea para la impresión en pantalla quedaría así:

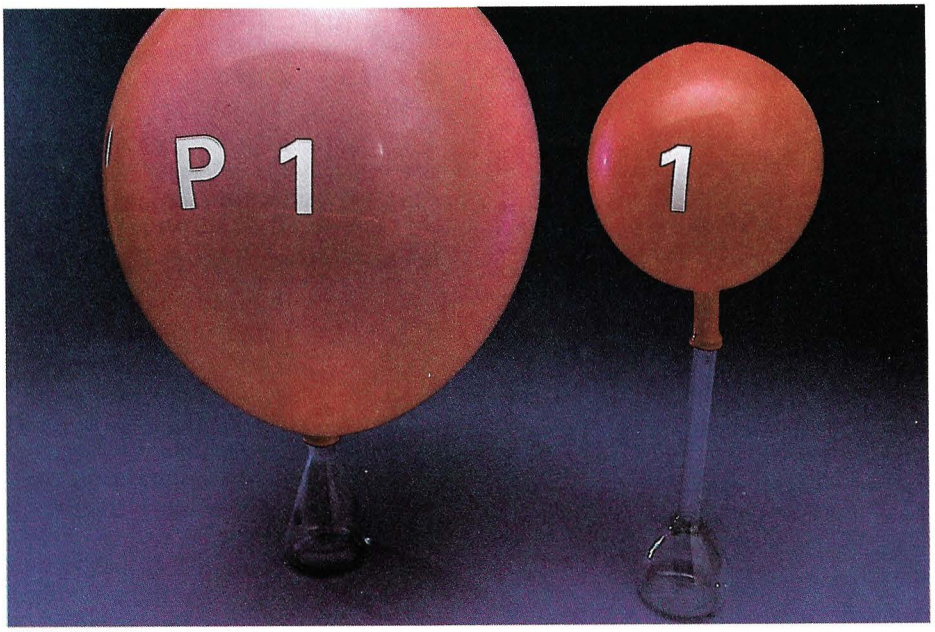
```
20 PRINT N$;TAB (10);B$+T$
```

El cálculo de los espacios en blanco es el mayor problema. En primer lugar se ha de conocer el máximo de caracteres que va a tener la cadena T$. En este caso se tratará de siete. Así pues, se reservará un número de espacios superior a siete, por ejemplo nueve. Por lo tanto, la cadena resultante de la operación B$+T$ ha de tener un total de nueve caracteres. Esto mismo se expresaría en BASIC como LEN(B$+T$)=9. Lo que interesa averiguar es la longitud que ha de tener B$. Esta longitud ha de ser tal que sumada a la longitud de T$ dé un total de 9. Ello significa que la longitud de B$ ha de ser igual a 9 menos la longitud de T$. En BASIC, tal valor se calcula como 9-LEN(T$). Sabiendo la cantidad de espacios se puede hacer uso de la función SPACE$ para generar una cadena de esa longitud.
En definitiva, la línea de programa encargada de calcular S$ quedará como se muestra a continuación.

```
17 LET S$=SPACE$(9-LEN(T$))
```

Con esta última línea se completa la rutina de presentación en pantalla de la lista de nombres y teléfonos. Las cinco líneas juntas ofrecerán el siguiente aspecto:

```
10 FOR I=1 TO 5
15 LET T$=STR$(T)
17 LET S$=SPACE$(9-LEN(T$))
20 PRINT N$;TAB (10);B$+T$
30 NEXT I
■
```



La presentación de los datos merece un especial cuidado. Para definir sus características de representación, el BASIC dispone de una versátil herramienta: el comando PRINT USING.

La ejecución de esta rutina conducirá al resultado deseado: los datos dispuestos en dos columnas, con los números justificados a la derecha:

```
RUN
MANOLO      2223344
LUIS        2341100
PEDRO       4442233
CARLOS       221122
JUAN        4005544
■
```

Sin embargo, este mismo resultado puede obtenerse por medio de un nuevo comando: PRINT USING. Este permite una gran variedad de formatos en la presentación de datos por pantalla o impresora.

PRINT USING: FORMATOS PARA DATOS NUMERICOS

La función USING se emplea en conjunción con el comando PRINT para especificar el formato en el que se van a presentar los datos. Dicho formato se codifica a continuación de la palabra USING, y afecta al resto de los datos contenidos en el argumento de la instrucción PRINT.
El formato se ha de indicar por medio de una cadena de caracteres que ha de ajustarse a ciertas reglas específicas.
En el caso de datos numéricos, se elige el número de dígitos a visualizar. Por cada posición se escribe un signo de número (#); como quiera que se trata de una cadena, dichos signos irán encerrados entre comillas. A título de ejemplo, se muestra una rutina que realiza la misma función comentada en el apartado anterior.

```
10 FOR I=1 TO 5
20 PRINT N$(I);TAB (10);USING
   "#########";T(I)
30 NEXT I
```

En la línea 20, la zona USING "#########" especifica que se deben reservar nueve

PRINT USING

Presenta en pantalla los datos usando el formato especificado como una cadena de caracteres.

Formato: PRINT USING <formato>; <dato1>;[<dato2>...]
Ejemplos: 20 PRINT USING "###.#"; 127.45
70 PRINT USING "!";A$;B$

espacios (tantos como símbolos "#" haya) para la representación del siguiente dato. Si el número tiene menos cifras, se rellenan los espacios sobrantes a la izquierda con "blancos". Esta rutina daría el siguiente resultado por pantalla:

```
RUN
MANOLO      2223344
LUIS        2341100
PEDRO       4442233
CARLOS       221122
JUAN        4005544
■
```

Como se observa, el resultado es idéntico al conseguido con el manejo de cadenas. La diferencia radica en la simplicidad de esta última rutina: el número de líneas se reduce de 5 a 3.
Dentro del formato no sólo se puede especificar el número de dígitos. Cuando se trata de datos numéricos con parte fraccionaria, es útil a menudo fijar el número de decimales a representar. Con el fin de reservar un espacio fijo para las zonas entera y fraccionaria, se actúa de forma muy similar a la anterior. Por ejemplo, si lo que se desea es mostrar cinco dígitos, de los cuales dos serán los correspondientes a la parte decimal, se empleará la expresión siguiente:

PRINT USING "####.##"; . . .

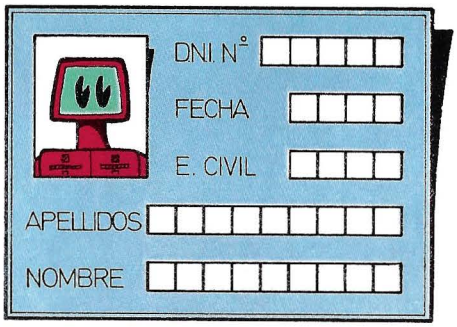
El punto decimal (notación anglosajona equivalente a la coma decimal), se coloca en el lugar correspondiente, entre signos "# ". Tomemos, por ejemplo, los siguientes números:

12.132 ,502.3 ,45.625 y 1208.332

Veamos qué ocurre al utilizar un formato como el anterior. El comando, ejecutado en forma directa, será el siguiente:

```
PRINT USING "###.##";
   12.132,502.3,45.625,1208.332 <CR>
12.13          502.30
45.63         %1208.33
■
```

Este ejemplo ilustra con claridad el funcionamiento del PRINT USING. En primera instancia, se reservan tantos espacios como signos "#" haya. Así, en el primer número (12.132), se deja un espacio antes del primer dígito. Ello se debe a que se han especificado 3 posiciones a la izquierda del punto decimal.
En cuanto a la parte fraccionaria, sólo se han impuesto dos dígitos; por lo tanto, sólo se representan las dos primeras cifras, truncándose las restantes. En la representación del tercer número (45.625) se ve que se redondean los decimales cuando hay truncamiento por la derecha.
Si el número a representar excede a la



El empleo de la alternativa PRINT USING permite especificar el formato en el que serán representados los datos en la pantalla, con plena comodidad y eficacia para el programador.

cantidad de posiciones especificadas, pueden ocurrir dos cosas. Pueden faltar posiciones a la derecha del punto decimal; en tal caso no se representan los dígitos sobrantes, redondeándose el último decimal. Si, por el contrario, los dígitos sobrantes se encuentran a la izquierda del punto, estos no se truncan. Si se truncara el número por la izquierda el error cometido sería muy grande, cosa que no sucede al redondear los decimales. Para avisar de la existencia de "desbordamiento", el BASIC imprime un signo de tanto por ciento en la primera posición de la presentación. Un ejemplo de ello lo constituye el último número visualizado en el ejemplo anterior.

FORMATOS CON SIGNO

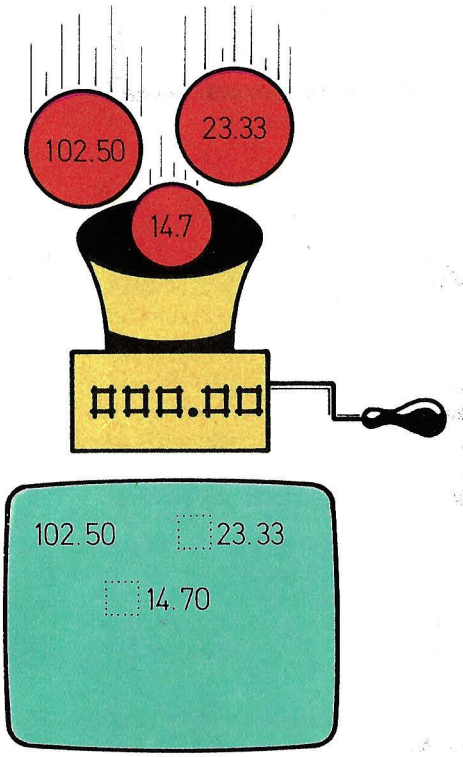
En el tipo de formato numérico anteriormente comentado, no se ha tenido en cuenta el signo (+ o −) del dato. Cuando el número es negativo, el signo (−) se representa consumiendo una de las posiciones reservadas. Véase el siguiente ejemplo:

```
10 PRINT USING "###";45
20 PRINT USING "###";-45
30 PRINT USING "###";145
40 PRINT USING "###";-145
```

La ejecución de estas cuatro líneas muestra cómo el signo ocupa una de las posiciones que se reservan en el formato. Este sería el resultado:

```
RUN<CR>
 45
-45
 145
%-145
■
```

Así pues, cuando se prevean números negativos, es aconsejable dejar una posición



Los signos de valor numérico (#) definen el número de espacios a reservar para la representación de los datos.

adicional para la representación del signo. El signo positivo, en cambio, no se representa. Si se desean visualizar los números con su respectivo signo (tanto positivo como negativo) habrá que especificar tal circunstancia en el formato. Ello se consigue incluyendo un carácter "+" en el mismo. En el caso del ejemplo precedente, esta exigencia se resolvería de la siguiente forma:

```
PRINT USING "¤¤";1985
```

El signo de tanto por ciento (%) aparece en pantalla dando a conocer que el dato excede el número de posiciones reservadas para su visualización.

```
10 PRINT USING "+###";45
20 PRINT USING "+###";-45
30 PRINT USING "+###";145
40 PRINT USING "+###";-145
```

Ahora todos los datos se representarán con su correspondiente signo:

```
RUN<CR>
 +45
 -45
+145
-145
■
```

Además del signo "+", los números positivos revelan otra diferencia respecto al ejemplo anterior. En efecto, el último dato no queda precedido por un signo de desbordamiento (%). La razón estriba en que el signo "+" del formato reserva una posición más. En los dos primeros datos se distingue esa posición en blanco a la izquierda del signo.
El signo (positivo o negativo) puede ser situado a la izquierda como a la derecha del dato. Para ello basta con colocarlo, dentro del formato, en la posición adecuada. A continuación, se muestra el mismo ejemplo pero con el signo a la derecha.

```
10 PRINT USING "###+";45
20 PRINT USING "###+";-45
30 PRINT USING "###+";145
40 PRINT USING "###+";-145
```

Con este nuevo formato se obtendrá por pantalla el siguiente resultado:

```
RUN<CR>
 45+
 45-
145+
145-
■
```

El comando PRINT USING permite también la representación del signo de otra forma. Se trata de representarlo a la derecha, pero únicamente cuando el dato sea negativo. Si el dato a representar es positivo, su signo (+) no se verá ahora reflejado en la pantalla.
Esta última modalidad de presentación tiene lugar cuando se sustituye el signo "+" por el "−", colocado *a la derecha* en el formato. De nuevo se utiliza el mismo ejemplo para contrastar la diferencia con anteriores formatos.

```
10 PRINT USING "###−";45
20 PRINT USING "###−";−45
30 PRINT USING "###−";145
40 PRINT USING "###−";−145
```

Y a continuación se presenta su resultado en pantalla. Compárese con el anterior.

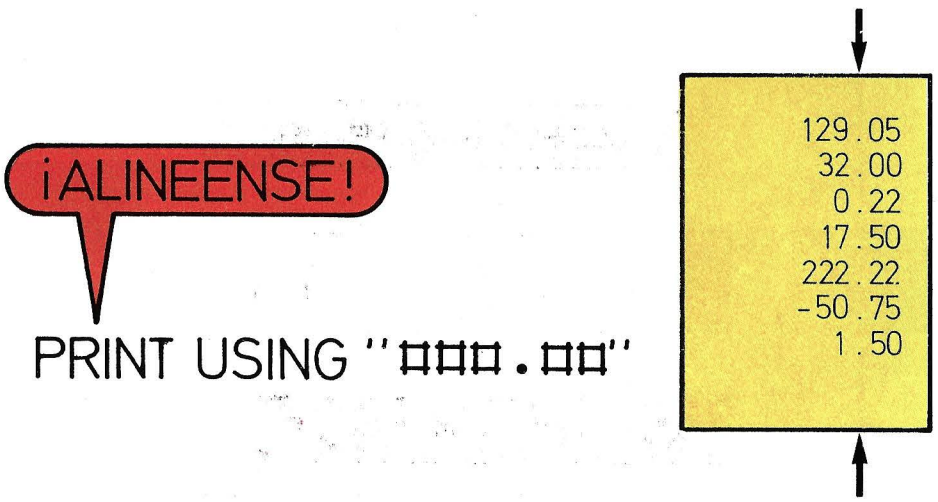
```
RUN<CR>
 45
 45−
145
145−
■
```

TABLA DE OPCIONES DEL COMANDO PRINT USING

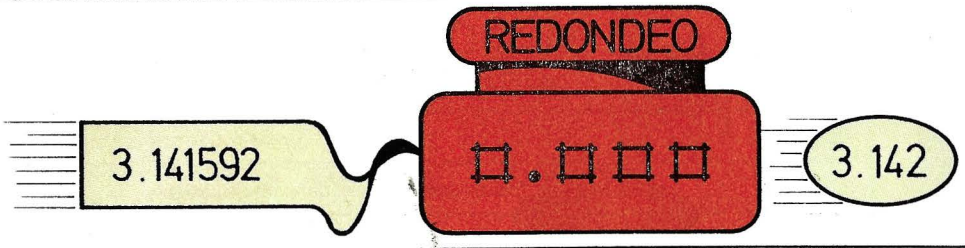
Cadenas	
!	Se muestra únicamente la inicial del dato de cadena.
\<blancos>\	Se representan tantos caracteres como blancos haya más dos. Si la cadena no tiene ese número de caracteres se justifica a la izquierda.
&	Se representa la cadena tal como es.
Números	
#	Se utiliza para especificar el número de dígitos a reservar.
.	Indica el lugar en el que se ha de situar el punto decimal.
+	Se presenta el signo del número, a la izquierda o a la derecha de éste.
−	Colocado a derecha exhibe el signo de los datos negativos.
**	Rellena los espacios en blanco, por la izquierda, con asteriscos.
$$	Muestra un signo "$" a la izquierda del primer dígito.
**$	Muestra un signo "$" a la izquierda del primer dígito, rellenando los espacios de más a la izquierda con asteriscos.
,	Sitúa comas cada tres dígitos a la izquierda del punto decimal.
^^^^	Fuerza la representación exponencial del dato.

OTROS SIMBOLOS EN EL FORMATO NUMERICO

Además de la representación del signo, existen otras marcas útiles a la hora de visualizar datos numéricos. Una de éstas es la coma, que en la notación anglosajona sirve para agrupar dígitos.
Como ya es sabido, en nuestro país (y en Europa en general) se acostumbra a separar la parte fraccionaria de la entera con una coma (2,5 en lugar de 2.5). De la misma forma, se suelen separar las unidades de millar de las centenas con un punto (1.000 representa lo mismo que 1000: mil). La notación anglosajona, por el contrario, emplea el punto para separar los decimales y la coma para agrupar los dígitos. El lenguaje BASIC, por su origen anglosajón, adopta la segunda alternativa.
En la cadena de formato que sigue al PRINT USING, se puede indicar la agrupación de dígitos con comas (notación anglosajona, como ya se ha indicado). Para ello, el carácter a incluir en dicho formato es una coma, situada en cualquier posición a la izquierda del punto decimal. Véase el siguiente ejemplo:



El uso de PRINT USING facilita la representación de listas de números. Todos ellos aparecerán con el punto decimal situado en la misma columna de la pantalla.



Los números cuya parte decimal contiene más dígitos de los especificados en el formato, son truncados por la máquina. El último dígito visualizado se redondea por la acción de este truncamiento.

```
10 PRINT USING "####.#";2222
20 PRINT USING "###,#.#";2222
```

En la primera línea no se ha incluido el carácter indicador de agrupamiento, mientras que la línea 20 sí lleva en su interior dicho carácter (la posición de éste podría haber sido cualquier otra *a la izquierda* del punto decimal). El resultado de su ejecución revela el efecto producido por la presencia del carácter "coma".

```
RUN<CR>
2222.0
2,222.0
■
```

Hay otros dos caracteres que producen un efecto específico en la representación de números. El primero es el signo de dólar ($). Dos de estos signos colocados a la izquierda del formato, harán que aparezca dicho carácter junto al dato representado. Este se visualizará inmediatamente a la izquierda del primer dígito.

```
PRINT USING "$$###";289<CR>
 $289
PRINT USING "$$###";7<CR>
   $7
■
```

El otro carácter especial es el asterisco (*). Una pareja de asteriscos colocada a la izquierda, producirá el efecto de rellenar los espacios en blanco con dicho carácter. Los espacios a rellenar son los que sobren al aplicar el formato especificado.

```
PRINT USING "*****";289<CR>
 **289
PRINT USING "*****";7<CR>
 ****7
PRINT USING "*****";1237<CR>
 *1237
■
```

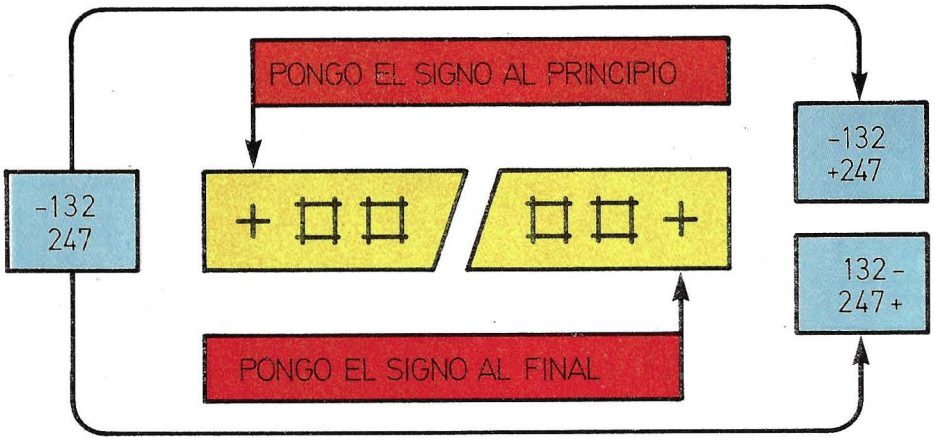
Como se puede observar, estos dos caracteres, al igual que el de signo, reservan también espacio para dígitos.
Se puede mezclar la acción de estos dos caracteres, logrando el efecto que se muestra en los siguientes ejemplos.

```
PRINT USING "**$##";289<CR>
 *$289
PRINT USING "**$##";7<CR>
 ***$7
PRINT USING "**$##";1237<CR>
 $1237
■
```

Para representar números de magnitud extrema, muy grandes o muy pequeños, se suele echar mano de la representación exponencial. En el formato del PRINT USING se puede especificar dicha modalidad sin más que colocar cuatro signos de exponenciación (^^^^) a la derecha de los de reserva de dígitos. Estos cuatro caracteres sirven para dejar otras tantas posiciones: una para el carácter "E", una para el signo y dos para los dígitos del exponente.

```
PRINT USING "#.#^^^^";1237<CR>
 1.2E+03
■
```

Todos los caracteres de indicación de formato pueden ser mezclados con entera libertad. La única salvedad está en los de representación de signo: en un formato sólo puede estar presente uno de estos caracteres.



La presencia del carácter "+" en el formato hará que se imprima el signo del dato; éste puede situarse tanto al principio como al final del mismo.

TABLA DE CONVERSION

ORDENADOR	PRINT USING	CADENAS		NUMEROS		
	PRINT USING	"\<blancos>\"	"&"	, (coma)	$$	**$
APPLE II (APPLESOFT)	—	—	—	—	—	—
APRICOT (M-BASIC)	PRINT USING	"\<blancos>\"	"&"	, (coma)	$$	**$
ATARI	—	—	—	—	—	—
CBM 64	—	—	—	—	—	—
DRAGON	PRINT USING	"%<blancos>%"	—	—	$$	**$
EQUIPOS MSX	PRINT USING	"\<blancos>\"	"&"	, (coma)	££	** ££
HP-150	PRINT USING	"\<blancos>\"	"&"	, (coma)	$$	**$
IBM PC	PRINT USING	"\<blancos>\"	"&"	, (coma)	$$	**$
MPF	—	—	—	—	—	—
NCR DM-V (MS-BASIC)	PRINT USING	"\<blancos>\"	"&"	, (coma)	$$	**$
NEW BRAIN	—	—	—	—	—	—
ORIC	—	—	—	—	—	—
SHARP MZ-700 (MZ-BASIC)	PRINT USING	"&<blancos>&"	—	(1)	££	—
SINCLAIR QL	—	—	—	—	—	—
SPECTRAVIDEO	PRINT USING	"\<blancos>\"	—	, (coma)	$$	**$
ZX-SPECTRUM	—	—	—	—	—	—

(1) Las comas deben situarse en el lugar real, es decir, cada tres dígitos.

FORMATEADO DE DATOS NO NUMERICOS

A la hora de representar datos alfanuméricos o de cadena de caracteres también puede especificarse un formato con PRINT USING. Se puede indicar el número de caracteres a representar, al igual que las posiciones de los dígitos en datos numéricos. Aquí no se utiliza el signo de número (#), sino que se recurre a espacios en blanco. Dichos espacios deben ir escoltados por "barras invertidas" (\), signo éste que no debe confundirse con el de división (/).

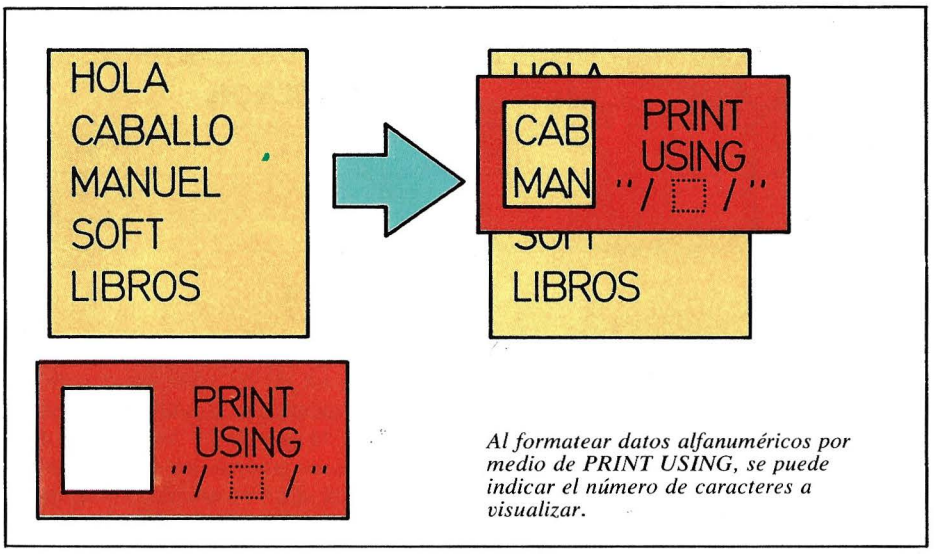
De la misma forma que los caracteres anteriores, éstos reservan dos posiciones más. Es decir, la cadena de formato "\ \" (<barra><espacio><barra>) indica que se van a representar *tres* caracteres. Estos caracteres serán los primeros (los de la izquierda) de la cadena a visualizar. Si la longitud de la cadena es menor que el espacio reservado, se rellenan los espacios sobrantes (a la derecha) con blancos. Véase un ejemplo:

```
10 A$="MI":B$="PERRO"
20 PRINT USING "\ \";A$;B$
30 PRINT USING "    ";A$;B$
```

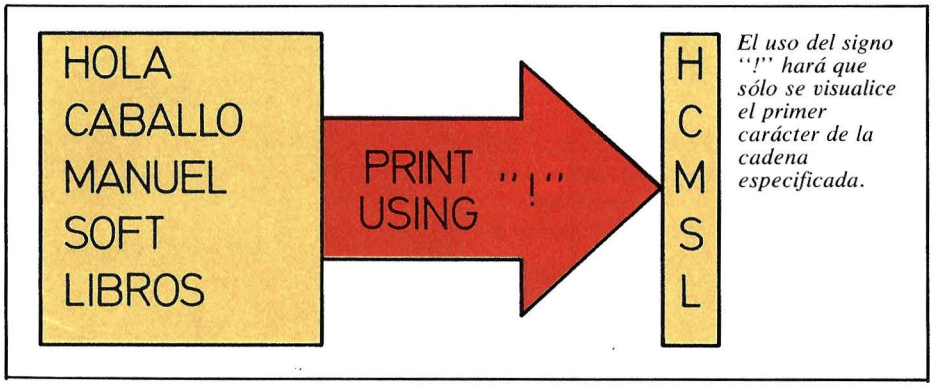
En la línea 20 no se ha dejado ningún blanco entre las dos barras inclinadas, luego se reservan dos espacios. La línea 20 incluye dos espacios entre barras, ello especifica la presentación de cuatro caracteres. El siguiente sería el resultado de ejecutar este pequeño programa:

```
RUN<CR>
MIPE
MI PERRO
■
```

Se puede apreciar que con este método, el número mínimo de caracteres a exhibir es de dos (uno por barra). Para representar un solo carácter se ha de variar el for-



Al formatear datos alfanuméricos por medio de PRINT USING, se puede indicar el número de caracteres a visualizar.



El uso del signo "!" hará que sólo se visualice el primer carácter de la cadena especificada.

mato. En este caso se pone simplemente un signo de admiración (!), éste indica que se mostrará tan sólo la inicial del dato.

```
20 PRINT USING "!";A$;B$
RUN<CR>
MP
MI PERRO
■
```

Por último, se dispone de un indicador adicional para visualizar el dato en toda su longitud. Se trata del signo "&". La utilización de este signo equivale a suprimir la incidencia de la partícula USING en el argumento de PRINT.
A diferencia de lo que ocurre con los datos numéricos, aquí no está permitido unir varios indicadores de formato. La razón es muy simple: especifican acciones incompatibles.

USO AVANZADO DE PRINT USING

El formato con el que se va a presentar el dato se especifica por medio de una cadena de caracteres especiales. Esta cadena no tiene, necesariamente, por qué ser un dato fijo. Puede utilizarse una variable para determinar el formato. El siguiente es un ejemplo de formato variable.

```
10 LET A$="**#.##"
20 PRINT USING A$;10
```

En él se hace uso de la variable de cadena A$ para almacenar el formato de presentación. Este es el resultado de ejecutar las dos líneas precedentes:

```
RUN<CR>
 *10.00
■
```

Utilizando variables para el almacenamiento de formatos, es posible que sea el propio programa quien elija el formato, de entre los almacenados en variables. También pueden generar distintos formatos empleando las funciones para el tratamiento de cadenas.
En el formato pueden incluirse caracteres distintos de los comentados. Estos no producirán más efecto que el de aparecer en pantalla. Un posible formato, conteniendo caracteres no estándar es el que muestra la siguiente pantalla:

```
10 PRINT USING "NUMERO**###";99
RUN<CR>
NUMERO***99
■
```

También es posible especificar formatos para más de un dato, aunque sean de distinto tipo. En el siguiente ejemplo se crea un formato para un dato de cadena y otro de tipo numérico.

```
10 PRINT USING "\ \**###";"NUMERO";99;
   "TE";121212
RUM<CR>

NUME***99TE %121212
■
```

En este último caso, es obligado situar los datos en el orden correcto; de otro modo, se produciría un error.

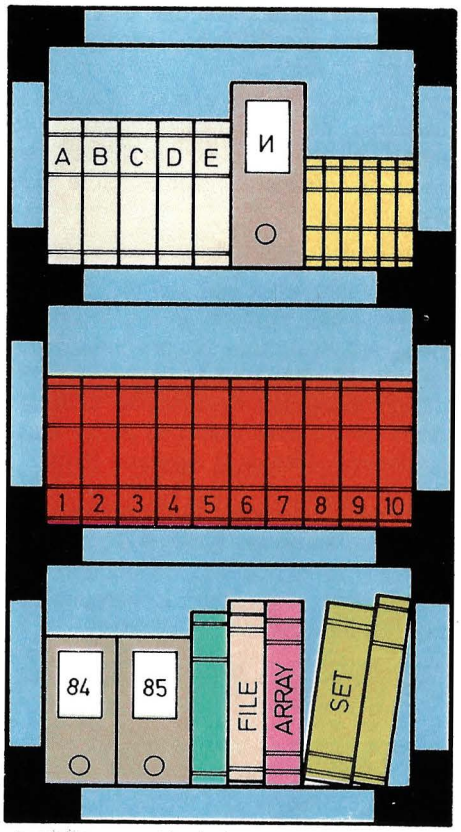
Pascal (9)

Ventajas de la programación estructurada en el manejo de los datos

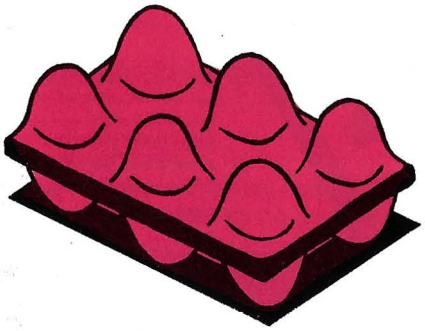
El fin primordial de la ciencia informática es el tratamiento de la información. Esta información toma cuerpo en los datos, que serán manipulados de acuerdo a los algoritmos apropiados para resolver cada problema específico.

En definitiva, los datos forman la componente estática del problema, mientras que la componente dinámica está constituida por los algoritmos que los manejan. En algunos casos, los datos serán fáciles de manipular y, por lo tanto, la componente dinámica del programa será también sencilla. Aunque la mayor parte de las veces no será así, sino que resultará obligado establecer algún tipo de relación entre los datos, que facilite su manejo y que simplifique la tarea de programación. Esto, ni más ni menos, hace patente la necesidad de estructurar los datos. Se define a una "estructura de datos", como una colección organizada de datos, en la que éstos están relacionados entre sí de alguna forma. Tal y como ocurría con la estructuración del programa, la estructuración de los datos presenta varias ventajas que justifican el estudio en profundidad de los datos a manejar en un programa. Entre otras contrapartidas, una estructura de datos bien pensada reduce enormemente la complejidad del programa que los va a manejar; por lo demás, aumenta la claridad de lectura y, en definitiva, mejora su seguimiento y facilita su posible modificación futura. Aún cabe citar otra razón más que aboga por el uso de datos estructurados: se consigue un mejor rendimiento al ejecutar el programa, debido a la menor cantidad de memoria necesaria para el almacenamiento de la información a procesar.

Dentro de sus virtudes como lenguaje de programación estructurado, el PASCAL permite la definición de cuatro tipos de datos estructurados: matrices (ARRAY), conjuntos (SET), registros (RECORD) y archivos (FILE).



La estructuración de los datos facilita el tratamiento de los mismos y confiere una mayor claridad al programa.



La estructura de tipo matriz está compuesta por un conjunto de elementos, todos ellos del mismo tipo.

MATRICES O "ARRAYS"

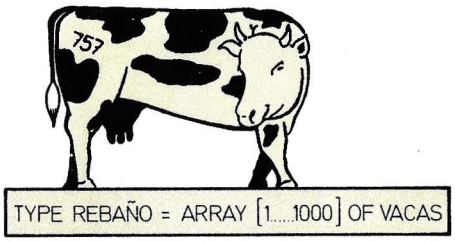
Las matrices deben su existencia en los lenguajes de programación, a la notación matemática que permite referirse a un conjunto de variables mediante un único vector de identificación y seleccionar un valor para ese vector mediante el empleo de subíndices.

Una matriz, se define pues, como un conjunto finito de componentes, todos del mismo tipo, que están ordenados en relación unívoca con un conjunto —también ordenado— de números enteros. Este conjunto de enteros ordenados recibe el nombre de subíndices. Cada uno de ellos permitirá el acceso a su componente correspondiente de la matriz.

Esta misma idea es extensible a matrices n-dimensionales, en las que cada componente será referenciado por más de un subíndice. Así, para el caso particular de una matriz de dos dimensiones cuyo nombre sea "Z", se tiene:

$$Z=\begin{bmatrix} z11 & z12 & \dots & z1n \\ z21 & z22 & \dots & z2n \\ \dots & \dots & \dots & \dots \\ zm1 & zm2 & \dots & zmn \end{bmatrix}$$

Cada componente es identificado por una expresión del tipo Z[i, j], en donde el subíndice "i" indica la fila y el subíndice "j" indica la columna. En este caso, los subíndices i y j podrán adoptar los valores enteros comprendidos entre 1 y m, y entre 1 y j, respectivamente; con lo que cada componente queda unívocamente determi-



Los elementos de un "array" están ordenados de acuerdo a su relación unívoca con un determinado subconjunto de los números enteros; esta relación es la que permite identificar a cada elemento.

nado. En PASCAL, el tipo de dato matriz se define con la siguiente notación:

TYPE <identificador>=ARRAY [<tipo indice>] OF <tipo base>;

Por ejemplo:

```
TYPE tabla=ARRAY [1..100] OF REAL;
     caracter=ARRAY [CHAR] OF
     INTEGER;
```

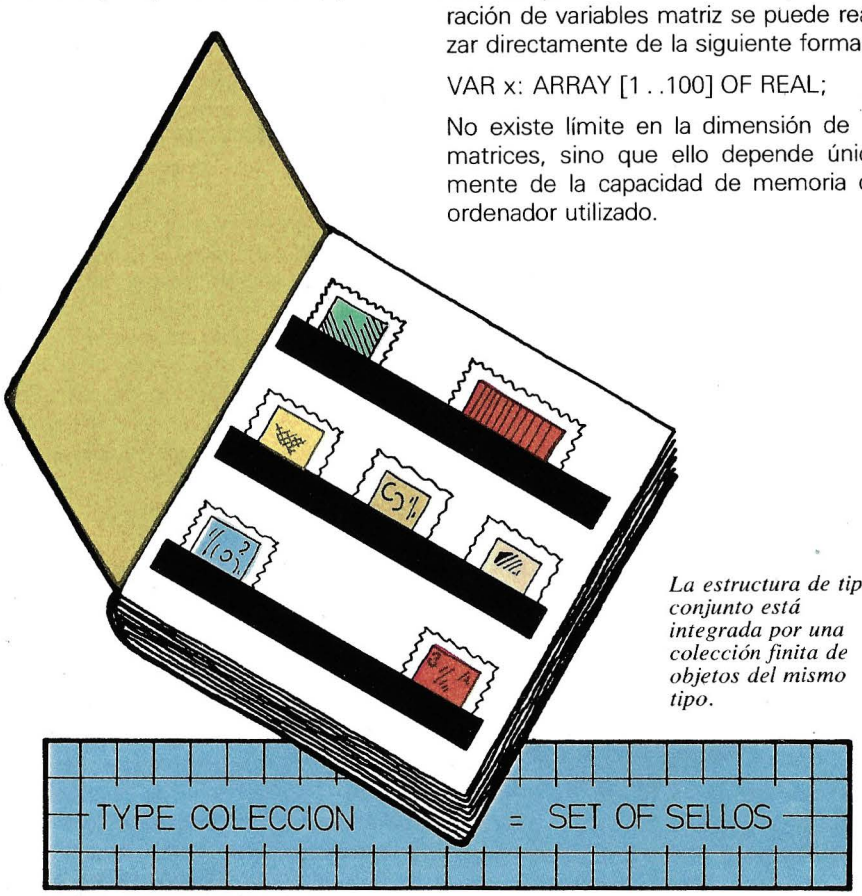
En la definición cabe distinguir varias zonas. El <identificador>, coincidente con el nombre del nuevo tipo de dato definido. El <tipo índice>, que puede ser cualquier tipo ordinal (excepto el INTEGER) previamente definido (CHAR, BOOLEAN, enumerado o subrango); éste fijará el número de elementos que formarán la matriz. Hay que señalar que en el ejemplo de la declaración del tipo "tabla", aunque el tipo del índice pueda parecer que es INTEGER, por estar formado por los números enteros del 1 al 100, no es así, sino que se trata de un tipo subrango de los enteros. Finalmente, hay que hablar de la zona <tipo base>; ésta revela el tipo de todos y cada uno de los componentes (ya que todos son del mismo tipo), y puede ser cualquiera, incluso otro tipo estructurado.
Es importante recordar que esta definición de tipo con TYPE, no declara variable alguna; en consecuencia, para poder utilizar este tipo de dato estructurado en un programa, habrá que declarar con VAR una variable matriz de un tipo ARRAY previamente definido. Por ejemplo, si se hace la declaración:

VAR x: tabla;

la nueva variable x será del tipo "tabla", que había sido definido en el ejemplo anterior. Así, la variable "x" quedará creada como una matriz de números reales con 100 componentes. No obstante, la declaración de variables matriz se puede realizar directamente de la siguiente forma:

VAR x: ARRAY [1..100] OF REAL;

No existe límite en la dimensión de las matrices, sino que ello depende únicamente de la capacidad de memoria del ordenador utilizado.



La estructura de tipo conjunto está integrada por una colección finita de objetos del mismo tipo.

La declaración de matrices multidimensionales se hace de la siguiente forma:

VAR parrafo:ARRAY [1..80] OF ARRAY [1..20] OF CHAR;

O más abreviadamente:

VAR parrafo:ARRAY [1..80,1..20] OF CHAR

Como ya se indicó, a cada elemento de la matriz se accede por su nombre seguido de los subíndices encerrados entre corchetes; subíndices que incluso pueden ser expresiones. A los elementos de la primera declaración de párrafo se accederá, entonces, mediante :parrafo[i] [j]; mientras que a los de la segunda se hará con :parrafo[i,j]. Así, serán válidas las siguientes asignaciones, suponiendo declaradas las variables: "letra", "I" y "J":

```
parrafo[8,12]:='A';
letra:=parrafo[50,10];
IF parrafo[I,J]=' ' THEN I:=I+1 ELSE
letra:='A';
```

Para aclarar mejor los conceptos relativos al uso de matrices de datos, veamos un sencillo ejemplo: un programa que realizará la multiplicación de dos matrices bidimensionales de números enteros (ver cuadro adjunto).
La operación de multiplicar matrices es específica de este tipo de datos, si bien se apoya en el producto y la suma de números enteros. De esta forma, el elemento [i,j] de la matriz resultante, se obtiene sumando los productos ordenados de los términos de la fila i-ésima de la primera matriz, por los términos de la columna j-ésima de la segunda matriz. Para ello, se dimensionan como constantes el rango de enteros que determinará el tamaño de las matrices, almacenándose éstas en las matrices A y B. A su vez, el resultado se almacenará en la matriz C.
Para multiplicar matrices de otras dimensiones, sólo habrá que modificar los valores constantes declarados con CONST.

EL TIPO SET

El lenguaje PASCAL permite manejar el concepto matemático de "conjuntos", ca-

```
PROGRAM multiplicarmatrices (INPUT, OUTPUT);
CONST M1=6; M2=4; M3=2
VAR I: 1 . .M1; J: 1 . .M3; K: 1 . .M2;
    dato: INTEGER;
    A: ARRAY [1 . .M1,1 . .M2] OF INTEGER;
    B: ARRAY [1 . .M2,1 . .M3] OF INTEGER;
    C: ARRAY [1 . .M1,1 . .M3] OF INTEGER;
BEGIN
      FOR I:=1 TO M1 DO
      BEGIN
            FOR K:=1 TO M2 DO
            BEGIN
                  READLN(dato);
                  A[I,K]:=dato
            END;
            WRITELN
      END;
      FOR K:=1 TO M2 DO
      BEGIN
            FOR J:=1 TO M3 DO
            BEGIN
                  READLN(dato);
                  B[K,J]:=dato
            END;
            WRITELN
      END;
      FOR I:=1 TO M1 DO
      BEGIN
            FOR J:=1 TO M3 DO
            BEGIN
                  DATO:=0;
                  FOR K:=1 TO M2 DO
                      dato:=dato+A [I,K]*B[K,J];
                      C[I,J]:=dato;
                      WRITE(dato)
            END;
            WRITELN
      END
END.
```

racterística ésta que no comparten la mayoría de los restantes lenguajes de programación. En todo caso, cabe señalar que su uso se ve reducido a casos muy particulares. Un conjunto se puede definir como: "una colección de elementos distintos entre sí, pero del mismo tipo".
Los conjuntos se representan en PASCAL escribiendo sus elementos entre corchetes:

[1,3,7,9,21,34,11]
[rojo,verde,azul]

Las variables de conjuntos, es decir, aquellas que se declaren como de algún tipo SET, tal y como se verá más adelante, pueden contener en un momento dado ninguno, uno, varios o todos los elementos que forman parte de dicho conjunto. Por lo tanto, habrá tantos posibles valores como resulte de elevar la base 2 a la potencia coincidente con el número de objetos que componen el conjunto.
El aspecto general de una definición del tipo conjunto viene dada por:
TYPE <identificador>= SET OF <tipo base>;

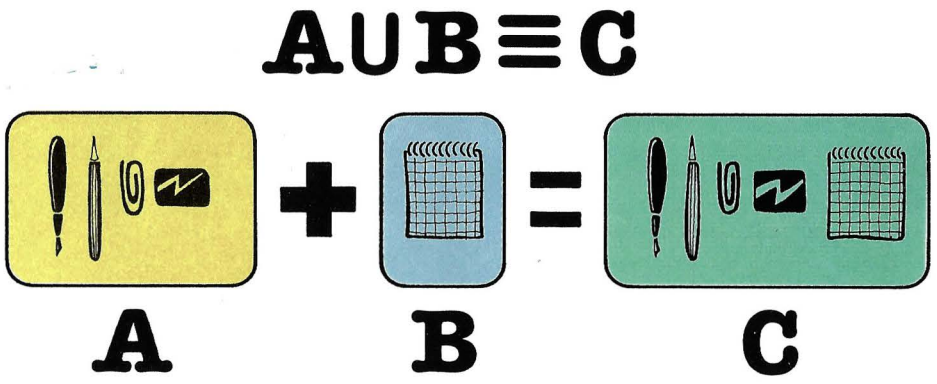
Por ejemplo:

```
TYPE numero= SET OF 0 . .9;
     palabra= SET OF CHAR;
```

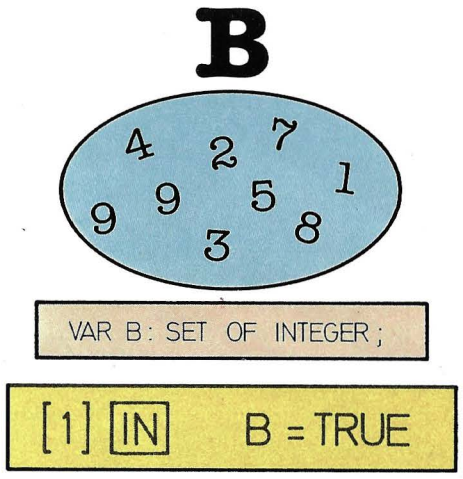
TABLA DE COMANDOS-PASCAL

COMANDO	COMETIDO	OBSERVACIONES
ARRAY [<tipo índice>] OF <tipo base>;	Definición del tipo matriz.	Zona de definición de tipos con TYPE.
SET OF <tipo>;	Definición del tipo conjunto.	Zona de definición de tipos con TYPE.
<conj.>*<conj.>	Operación de intersección de conjuntos.	Cuerpo del programa.
<conj.>+<conj.>	Operación de unión de conjuntos.	Cuerpo del programa.
<conj.>−<conj.>	Operación de diferencia de conjuntos.	Cuerpo del programa.
<elem.> IN <conj.>	Relación de pertenencia o inclusión en un conjunto.	Cuerpo del programa.
<conj.> <=<conj.>	Relación de inclusión entre conjuntos.	Cuerpo del programa.

NOTA: <tipo indice>: cualquier tipo ordinal, excepto INTEGER. <tipo base>: cualquier tipo de dato incluso otro estructurado. <tipo>: tipo de dato escalar o subrango, excepto REAL. <conj.>: conjunto de un SET. <elem.>: elemento de un SET.



El PASCAL permite realizar las operaciones típicas entre conjuntos; por ejemplo, la unión de conjuntos, cuyo resultado es un nuevo conjunto que agrupa a todos los elementos de los conjuntos afectados por la operación.



La función IN comprueba la relación de inclusión establecida entre conjuntos; esto es: verifica la pertenencia de un elemento a un conjunto.

El <tipo base> puede ser de cualquier tipo escalar o subrango, excepto de tipo REAL. Como siempre, la definición de tipo no declara la variable, por lo que ésta debe declararse aparte, o bien directamente:

```
TYPE colores=(rojo,verde,azul,amarillo,
blanco,negro);
VAR cuadro: SET OF colores;
```

Cada elemento de tipo SET se referencia por medio de un subrango de valores: [1 . .5], o bien por un subconjunto de esos valores separados por comas: [1,2,3,4,5]. Para no complicar en exceso el cálculo de valores por parte del compilador, ni ocupar excesiva memoria, y debido al uso esporádico de este tipo de datos, se suele limitar el número de elementos del conjunto. Un valor típico se concreta en un máximo de 64 elementos.

Para este tipo de datos existen una serie de operadores específicos. Estos realizan las operaciones básicas entre conjuntos definidas por la teoría matemática moderna. Así, para la unión de conjuntos se emplea el símbolo '+', para la intersección el símbolo '*', y para la diferencia el símbolo '−'. Las siguientes operaciones darán como resultado el que se indica:

```
[azul,rojo]+[rojo,verde]=[azul,rojo,verde]
[azul,rojo]*[rojo,verde]=[rojo]
[azul,rojo]−[rojo,verde]=[azul]
```

Además de los operadores definidos, se pueden aplicar a elementos de tipo SET los operadores de relación ya conocidos: =, < >, <=, >=. Estos últimos darán como resultado un valor lógico TRUE o FALSE.

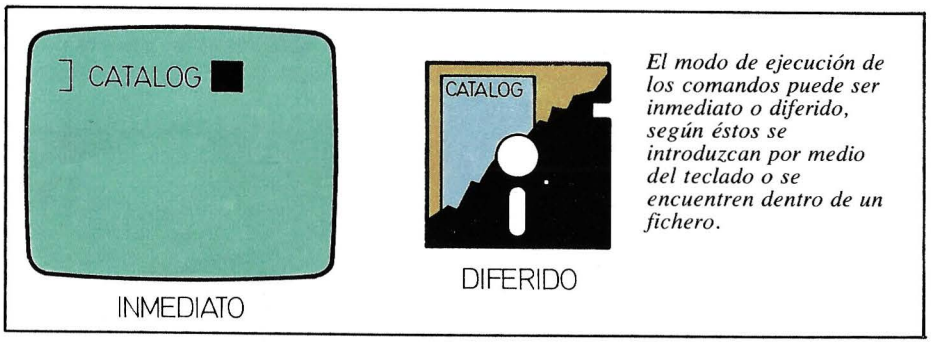
Los operadores: '<=' y '>=' no tienen el sentido de comprobar si un conjunto es mayor o menor que otro, ya que ello es absurdo, sino que tienen la función de comprobar si un conjunto está o no incluido en otro. Así, por ejemplo: la expresión lógica [verde]<=[azul,verde], dará como resultado TRUE al ser evaluada. Para comprobar la pertenencia o inclusión de conjuntos existe un operador específico. Este es el operador binario 'IN', que obtiene un resultado de tipo lógico (TRUE o FALSE). Así, la expresión: [verde] IN [rojo,azul] adoptará el valor FALSE, mientras que: [verde] IN [rojo,verde] será TRUE.

Los tipos RECORD y FILE, por su importancia, se estudiarán con más detalle y detenimiento en el próximo capítulo de la obra.

Apple DOS 3.3 (2)

Comandos de acceso a disco y control del entorno

El mantenimiento y la transformación de las informaciones almacenadas en las distintas unidades de disco del sistema, ya se trate de programas o de datos, exige un lenguaje sencillo, y a la vez potente, con el cual gestionar dicha información. Dicha función es realizada por los comandos del sistema operativo; éstos indican la acción a realizar, además del objeto a quien afecta dicha acción.



El modo de ejecución de los comandos puede ser inmediato o diferido, según éstos se introduzcan por medio del teclado o se encuentren dentro de un fichero.

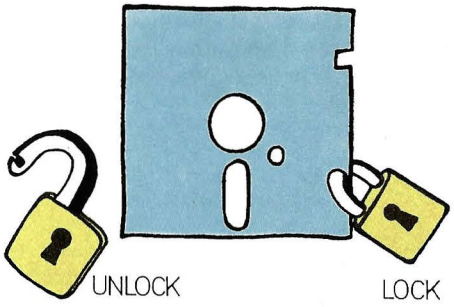
SINTAXIS

Todos los comandos del sistema operativo DOS 3.3 están construidos según unas reglas de formación fijas y preestablecidas; reglas que hay que respetar estrictamente para que el sistema operativo entienda, sin ambigüedades, las instrucciones que hacia él se dirigen.
Al igual que ocurre en el vocabulario de cualquier otro idioma, los comandos del sistema operativo tienen una sintaxis propia, aunque extremadamente sencilla. Dicha sintaxis dicta que todos los comandos han de expresarse de la siguiente forma, no siendo necesario especificar siempre todos los argumentos:

COMANDO [nf] [,Sn] [,Dn] [,Vn]

COMANDO representa a cualquier comando del DOS, mientras que las letras encerradas entre corchetes corresponden a los diversos argumentos aplicables al comando en cuestión.
Los argumentos se han representado entre corchetes para evidenciar que no es obligatorio incluirlos siempre, sino tan sólo en caso necesario. Cada uno de los argumentos representa a los siguientes conceptos:

nf: el nombre del fichero sobre el que se aplica el comando.
Sn: "n" coincide con el número de conector o "slot" (1 a 7) al que está conectada la tarjeta que controla a la referida unidad de disco.
Dn: "n" indica el número de unidad de disco (1 ó 2) que está controlada por una misma tarjeta.
Vn: "n" indica en este caso el número del volúmen (0 a 256) con que fue marcado el disquete en el instante de su inicialización.
Todas las letras contenidas en un comando han de figurar en mayúsculas para que sean reconocidas como tales por el sistema operativo.



El comando LOCK bloquea un archivo de tal forma que su contenido no puede ser alterado mientras se encuentre en dicho estado. Para desbloquearlo hay que utilizar el comando UNLOCK.

MODOS DE EJECUCION

Siempre que el sistema operativo tiene bajo su control a la pantalla y al teclado, es decir, cuando no se está ejecutando ningún otro programa, al ser teclada cualquier secuencia de caracteres seguidos por una acción sobre la tecla <ENTER> o <RETURN>, el sistema operativo compara los caracteres introducidos con la lista de comandos existentes. Si dicha secuencia de caracteres no está incluida en la mencionada lista, estos caracteres pasarán al traductor BASIC que esté activo en ese momento. El BASIC tratará de ejecutarlos como si fuera una sentencia de dicho lenguaje, devolviendo un mensaje de error en el hipotético caso de que estos caracteres tampoco tuviesen significado para él.
Los comandos se leen generalmente desde el teclado, aunque también pueden estar embebidos en un programa BASIC, o contenidos en un fichero de comandos.
Los comandos introducidos desde el teclado son ejecutados al instante; en tal caso, cabe afirmar que son comandos de ejecución inmediata. Sin embargo, si el comando está contenido en un fichero, ya

sea en un programa BASIC o en un fichero EXEC o de comandos, éste no será ejecutado hasta que se active el programa BASIC o el fichero EXEC, y llegue su turno de ejecución.

DESCRIPCION DE LOS COMANDOS

Los comandos que se describen a continuación hacen referencia al acceso a las unidades de disco del sistema, así como al entorno en el que operan. En capítulos posteriores se analizarán aquellos que se refieran al acceso a ficheros de acceso directo, secuencial y binarios.

- CATALOG

El efecto de este comando es listar por pantalla el número del volumen y el nombre de todos los ficheros contenidos en la unidad de disco especificada; mostrando, además, el tipo de cada fichero, su tamaño en sectores y si está o no bloqueado (circunstancia ésta indicada por la presencia o no de un asterisco delante del tipo de fichero).
Su sintaxis es:

CATALOG [,Sn] [,Dn]

- INIT

Organiza la superficie del disco en sectores y pistas, escribiendo ceros en todos los campos de datos. Una vez realizada esta tarea, el comando se encarga de incluir en el disco una copia del sistema operativo y un programa de bienvenida.

INIT [nf] [,Sn] [,Dn] [,Vn]

- DELETE

Borra del disco especificado el fichero cuyo nombre coincida con el indicado en el argumento, siempre y cuando dicho archivo se encuentre desbloqueado.

DELETE [nf] [,Sn] [,Dn] [,Vn]

- RENAME

Cambia el nombre del fichero indicado en nf1, adoptando como nuevo nombre el especificado en nf2. Ello no afecta en absoluto al contenido del fichero, siempre y cuando no exista previamente otro fichero con el mismo nombre que nf2.

RENAME nf1,nf2 [,Sn] [,Dn] [,Vn]

- LOCK

Protege los ficheros para que no sufran daños accidentales, bloqueando dicho archivo de manera que no puedan efectuarse sobre él las operaciones de actualización, borrado o cambio de nombre. Los archivos así protegidos se distinguen fácilmente por aparecer precedidos por un asterisco en el listado del catálogo.

LOCK nf [,Sn] [,Dn] [,Vn]

- UNLOCK

Desbloquea el fichero previamente protegido con el comando LOCK, posibilitando las operaciones de actualización, borrado y cambio de nombre del fichero.

UNLOCK nf [,Sn] [,Dn] [,Vn]

- VERIFY

Comprueba que un fichero puede ser leído desde el disco o la memoria del ordenador sin error alguno, debiendo estar dicho fichero desbloqueado.

VERIFY nf [,Sn] [,Dn] [,Vn]

- RUN

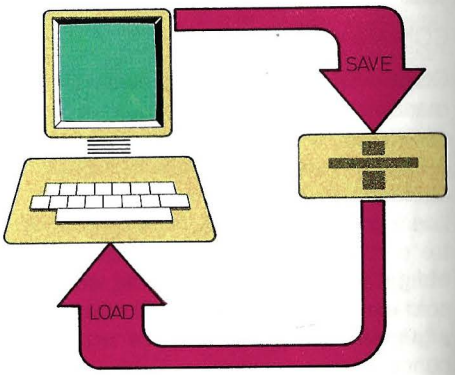
Copia un programa BASIC de tipo A o I, leyéndolo del disco y depositándolo en la memoria del ordenador; tras ello, lo ejecuta.

RUN nf [,Sn] [,Dn] [,Vn]

- LOAD

Copia un programa desde el disco a la

El DOS 3.3 de Apple es un sistema operativo casi exclusivamente orientado a la manipulación de los archivos en disco; para ello cuenta con un amplio repertorio de comandos especializados en las diversas funciones.



Los comandos LOAD y SAVE permiten transferir los programas desde el disco a la memoria central del ordenador, y viceversa.

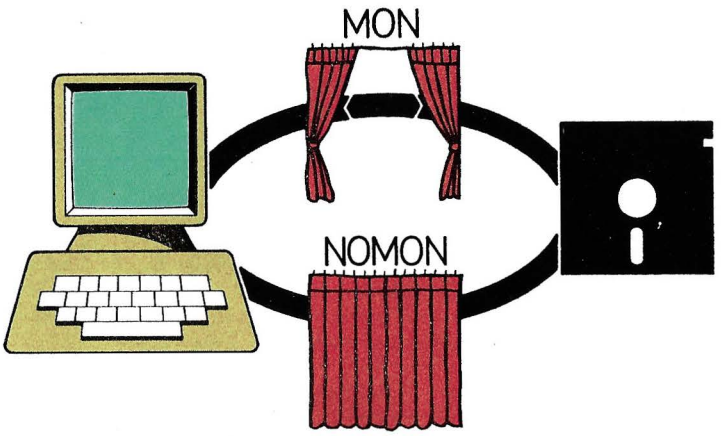
memoria primaria del ordenador, borrando el contenido anterior de la memoria. Al tiempo, activa el traductor BASIC que corresponda, según sea el tipo del fichero que contenía al programa (A para BASIC Applesoft, e I para Integer).

LOAD nf [,Sn] [,Dn] [,Vn]

- SAVE

Graba o traslada el programa que reside en la memoria a un fichero de disco, cuyo tipo vendrá determinado por el BASIC activo en ese instante.

SAVE nf [,Sn] [,Dn] [,Vn]



Mediante el comando MON se puede visualizar el flujo de información entre la unidad de disco y el ordenador; ello ayudará a revelar los problemas de funcionamiento.

Las redes locales

La filosofía de conexión de diversos tipos de ordenadores (micros, minis y grandes ordenadores) y de los distintos periféricos a ellos asociados (pantallas, trazadores, impresoras, unidades de almacenamiento masivo de datos, etc.), por medio de una línea que permita su intercomunicación en un radio de acción limitado, está abriéndose paso día a día. Cada vez es mayor su desarrollo, a la par que su aceptación, ya que permite una distribución eficaz de los recursos del sistema entre todos los componentes de la red.

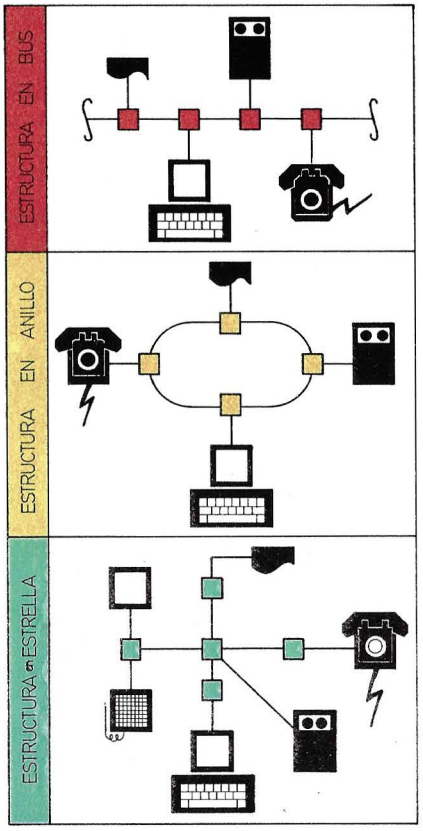
La idea está basada en la conexión de los diversos elementos que forman la red a través de una línea o canal de conexión y un interface de comunicaciones.

Entre los medios de transmisión más comúnmente utilizados se encuentra el simple par de hilos trenzados, del tipo empleado en las comunicaciones telefónicas; este medio se traduce en una baja velocidad de transmisión (de 300 a 9.600 Kbits/seg) y un acusado nivel de interferencias. El cable coaxial es otra de las soluciones más frecuentes, debido a su bajo nivel de interferencias y a su apreciable velocidad de transmisión (de 10 a 12 Mbits/seg).

Por último, se encuentra el cable de fibra óptica, constituido por dos materiales —uno con mayor índice de refracción que el otro— capaces de transmitir señales ópticas; con esta nueva técnica se consiguen capacidades de transmisión muy elevadas (de 150 a 2.500 Mbits/seg).

Los diversos modos de conexión entre los ordenadores llevan diferentes configuraciones típicas. La conexión en bus se basa en la integración a lo largo del medio de conexión de los diversos elementos de la red, no existiendo ningún elemento que actúe como centralizador de comunicaciones, y recogiendo cada estación sólo los mensajes a ella dirigidos (toda una solución para evitar aglomeraciones).

En la conexión en anillo, todos los elementos están asociados a una línea



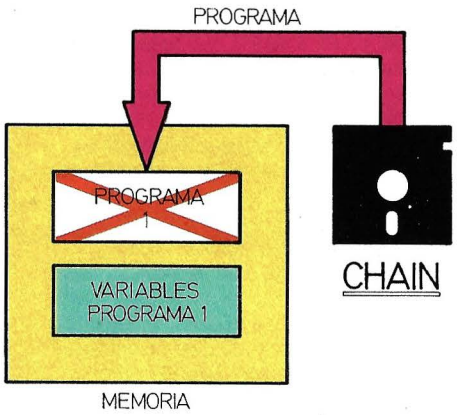
circular. Cada estación recibe todos los mensajes, si bien, retransmite los que no van dirigidos a ella; ello reduce obviamente la velocidad de transmisión, al tiempo que deja en pie el peligro de que se produzca un corte en la transmisión al estropearse alguna estación.

La conexión en estrella se produce cuando todos los elementos están conectados, uno a uno, a una estación central encargada de controlar las comunicaciones. Ello evita el inconveniente del posible corte de la transmisión por avería de un elemento, aunque grava la velocidad de transmisión en redes con mucho tráfico. Los métodos utilizados en la transmisión son el "token passing" y el CSMA/CD (Carrier Sense Multiple Access with Collision Detect).

En el primero, la información se transmite mediante un conjunto de bytes o "tokens" que circulan constantemente a través de la red. La información se introduce en aquellos que estén libres, indicando cuál es su destino; así, recorrerán una a una las estaciones de la red hasta llegar a su destino.

Por su parte, el método CSMA/CD se basa en la presencia permanente de una señal en el canal que indica si éste se encuentra ocupado o libre. En el caso de que esté libre, una estación puede mandar un mensaje por el canal, indicando ahora que el canal pasa a estar ocupado. Dado que la señal indicadora del estado del canal se desplaza a velocidad finita, puede que se transmitan dos mensajes casi simultáneamente. Tal situación haría que éstos colisionaran en el canal, en cuyo caso se suspenderán las transmisiones que serán reanudadas en un intervalo de tiempo aleatorio.

DOS 3.3 Apple II	
	Comandos de acceso a disco y definición del entorno
CATALOG	Muestra lista de ficheros del disco
INIT	Inicializa o formatea el disco
DELETE	Borra un archivo
RENAME	Cambia el nombre de un archivo
LOCK	Bloquea un archivo, pudiéndose acceder sólo para lectura
UNLOCK	Desbloquea un archivo
VERIFY	Comprueba si no hay error de lectura/escritura en un fichero
RUN	Ejecuta un programa
LOAD	Carga en memoria un programa desde el disco
SAVE	Graba en el disco el programa residente en la memoria
PR#	Indica el dispositivo de salida
IN#	Indica el dispositivo de entrada
FP	Activa el BASIC Applesoft
INT	Activa el BASIC Integer
MON	Permite visualizar la información enviada desde la memoria al disco
NOMON	Desactiva la capacidad anterior
EXEC	Ejecuta los comandos contenidos en un archivo
MAXFILES	Especifica el número máximo de archivos abiertos a la vez
CHAIN	Carga un programa sin borrar las variables anteriores (BASIC Integer sólo)



*El comando **CHAIN** permite conservar las variables del programa que anteriormente ocupaba la memoria interna; de esta forma, será posible utilizar sus valores dentro del nuevo programa que se cargue en memoria.*

- PR #n

Envía los caracteres que normalmente se escriben en pantalla al dispositivo conectado al "slot" o conector cuyo número es "n".

- IN #n

Lee caracteres desde el dispositivo conectado al "slot" n, en lugar de hacerlo desde el teclado.

- FP

Activa el BASIC Applesoft, cargándolo desde la tarjeta que lo contiene, o desde un disco en el caso de que no se disponga de dicha tarjeta.

FP [,Sn] [,Dn]

- INT

Activa el BASIC Integer.

- MON

Normalmente, el trasiego de información entre el disco y el microordenador no se muestra al usuario, con lo que el proceso resulta transparente de manera que no pueda confundir o molestar al operador. Sin embargo, a veces es aconsejable visualizar dicha información; por ejemplo, a la hora de buscar un error en un programa. El comando MON posibilita esta acción sirviéndose de sus argumentos:
C: Muestra todos los comandos de disco.
I: Muestra la información dirigida desde el disco al microordenador.
O: Muestra la información dirigida desde el microordenador al disco.
Su sintaxis es:

MON [C] [,I] [,O]

- NOMON

Desactiva la visualización de informaciones conseguida con el comando anterior.

NOMON[C] [,I] [,O]

- EXEC

Hace que el sistema operativo acepte comandos residentes en un fichero de comandos en vez de hacerlo desde el teclado.
El fichero de comandos puede contener datos, sentencias BASIC o comandos del sistema operativo. Para su ejecución es necesario indicar la primera línea del fichero en donde se desea comenzar la ejecución (argumento Rn).

EXEC nf [,Rn] [,Sn] [,Dn] [,Vn]

- MAXFILES n

Especifica el número máximo de ficheros que pueden estar abiertos a la vez (de 1 a 16).

- CHAIN

Carga un nuevo programa sin cerrar los archivos que estuviesen abiertos y sin destruir las variables del programa que residía anteriormente en memoria. Ello permite al nuevo programa operar con los resultados del programa previo. Sólo es aplicable a programas creados en BASIC Integer.
Sintaxis:

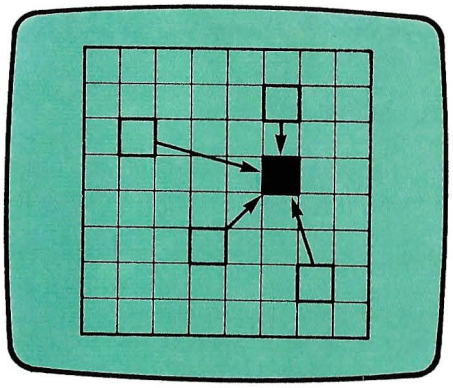
CHAIN nf [,Sn] [,Dn] [,Vn]

Multiplan (2)

Repertorio de funciones del Multiplan

Para finalizar el estudio teórico de la hoja electrónica MULTIPLAN, se detallan a continuación todas las funciones que este programa es capaz de manejar. Su uso está reservado para aquellas celdas de la hoja electrónica que deben ser calculadas en función del contenido de otras celdas. En resumen, si se acepta que el objetivo principal de una hoja electrónica es facilitar al usuario el cálculo de ciertos datos en función de otros, hay que concluir que un alto porcentaje de su rendimiento estará supeditado a la eficacia y variedad de las funciones disponibles.

FUNCIONES DEL MULTIPLAN



Una celda de la hoja electrónica se dice que es función de otras cuando el valor de aquélla se puede obtener a partir del contenido de éstas.

La hoja electrónica MULTIPLAN dispone de 40 funciones distintas que ponen en manos del usuario una considerable potencia de cálculo. Con objeto de ofrecer una visión organizada de las funciones del MULTIPLAN, vamos a clasificarlas en cuatro grandes grupos.

El primero integrará a todas las funciones matemáticas; grupo que, con pequeñas variaciones, coincide con el propio de otros programas de este tipo.

El segundo grupo de funciones recogerá a todas aquellas que manejan variables lógicas.

El tercer grupo, también tradicional en el terreno de las hojas electrónicas, lo integran las funciones dedicadas a cálculos financieros.

Y, por último, un cuarto grupo en el que caben algunas funciones especiales que pueden ser utilizadas en cualquier tipo de aplicación. Este último grupo es el que representa una diferencia respecto a otros programas, puesto que contiene funciones atípicas diseñadas por los creadores de MULTIPLAN.

FUNCIONES MATEMATICAS

1. ABS (N)

Tiene un único argumento N que debe ser numérico, y del que la función ABS devuelve su valor absoluto. Por ejemplo: ABS (17.5) corresponde al valor 17, mientras que ABS (R5C4) obtiene el valor absoluto del contenido de la celda situada en la 5.ª fila y la 4.ª columna de la hoja electrónica.

2. SIGN (N)

También dispone de un único argumento

N y, según sea su valor, la función SIGN devolverá tres posibles resultados: −1 si N es negativo, 0 si N es nulo, +1 si N es positivo. Ejemplos: SIGN (−15.10) = −1; SIGN (−10+10) = 0; SIGN (R5C4) = 1, si el número contenido en R5C4 es positivo.

3. SIN (N)

La función trigonométrica SIN calcula el seno del argumento N que debe indicarse en radianes. Por ejemplo: SIN(PI/2) = 1.

4. COS (N)

En esta ocasión, la función trigonométrica COS calcula el coseno del ángulo N medido en radianes. Por ejemplo: COS (PI/2) = 0.

5. TAN (N)

La tercera función trigonométrica del MULTIPLAN permite calcular la tangente de un ángulo N, cuya medida en radianes aparece en su argumento. Ejemplo: TAN(PI/3) = 1,732.

6. ATAN (N)

La función ATAN es la función inversa de TAN, es decir, calcula el ángulo medido en radianes cuya tangente vale N. En Matemáticas, esta función recibe el nombre de arcotangente. Por ejemplo: ATAN (1,732) = PI/3. Obsérvese cómo al aplicar la función ATAN al resultado del ejemplo anterior se obtiene, precisamente, el argumento de dicho ejemplo.

7. EXP (N)

El matemático Euler, a partir de densos estudios, llegó a deducir un número que tenía ciertas propiedades interesantes: 2,7182818... En su honor, a dicho número se le denomina número "e". La función EXP de MULTIPLAN calcula el valor del número "e" elevado al argumento N. Por ejemplo: EXP (1) = 2,7182818.

8. LN (N)

La función logaritmo Neperiano, LN, es la inversa de la función EXP. Ello se debe a que la base de LN es, precisamente, el número e. Por lo tanto, al aplicar LN al número e, el resultado obtenido será 1.

9. LOG 10 (N)

Sirve para calcular el logaritmo decimal del argumento N.
Ejemplo: LOG 10 (10) = 1. En general, al aplicar la función logaritmo a un argumento igual a su base, el resultado siempre es 1.

10. INT (N)

La función parte entera, como su propio nombre indica, devuelve como resultado el valor del argumento despreciando la parte decimal. Por ejemplo: INT (87.25) = 87; INT (2) = 2.

11. ROUND (N)

El objetivo de esta función es análogo al de la anterior; esto es: a partir de un argumento numérico con parte decimal, producirá un valor entero que coincidirá con la parte entera del argumento, siempre que la parte decimal sea inferior a 0.5. Si la parte decimal es superior a 0.5 el resultado coincidirá con la parte entera del número más uno. Esta función recibe el nombre de redondeo. Ejemplos: ROUND (17.9) = 18; ROUND (17.4) = 17.

12. SQRT (N)

La función raíz cuadrada SQRT se limita a devolver la raíz cuadrada de su argumento. Por ejemplo: SQRT (4) = 2.

13. MOD (N,M)

La función módulo MOD tiene dos argumentos N y M. El resultado producido es el resto de la división entera de N entre M. Por ejemplo: MOD (8,3) = 2. En general, se cabe afirmar que MOD (N,M) = N − INT (N/M) * M.

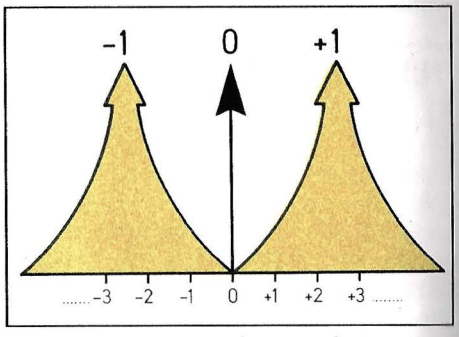
14. PI ( )

La función PI no tiene ningún argumento, ya que en realidad es una función constante que produce como resultado PI, es decir = 3,1415926...

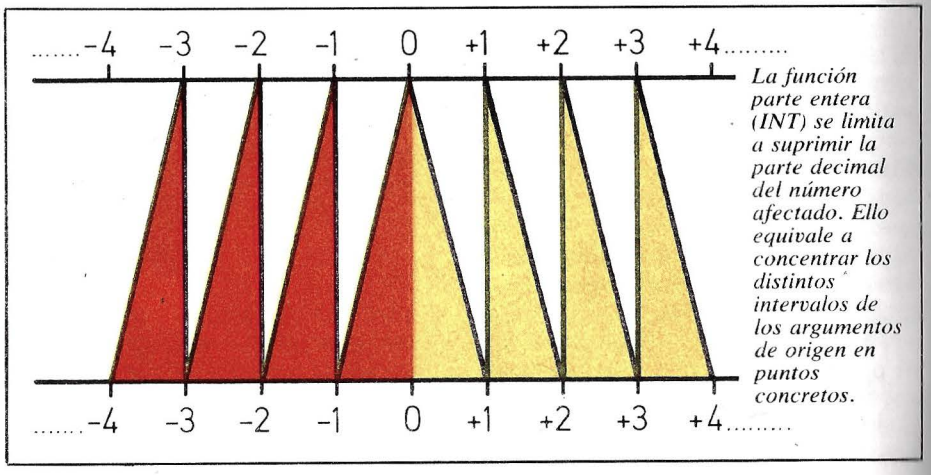
FUNCIONES LOGICAS

1. TRUE ()

La función lógica TRUE es constante y, precisamente por ello, no necesita ningún argumento. En cualquier caso devuelve como resultado el valor lógico VERDADERO.



La función SIGN transforma cualquier número negativo en el dato −1, cualquier número positivo en +1 y el cero en sí mismo.



La función parte entera (INT) se limita a suprimir la parte decimal del número afectado. Ello equivale a concentrar los distintos intervalos de los argumentos de origen en puntos concretos.

2. FALSE ()

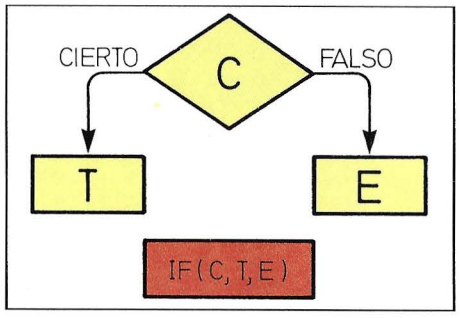
Esta función es la inversa de la anterior; es decir, al invocar la función FALSE, sin ningún parámetro, se obtendrá la constante lógica FALSO.

3. AND (lista de argumentos lógicos)

La función lógica AND admite como argumentos una lista de expresiones lógicas. Producirá como resultado el valor VERDADERO "si y sólo si" todas las expresiones de entrada tienen valor VERDADERO; en cuanto exista al menos una con valor FALSO, el resultado será FALSO. Por ejemplo: AND (12 + 5 = 17, 3 = 2 + 1) = VERDADERO, AND (5 = 3 + 3, 2 = 1 + 1) = FALSO.

4. OR (lista de argumentos lógicos)

Esta función lógica también admite como argumento una lista de expresiones lógicas. El resultado producido será en este caso VERDADERO cuando alguna de ellas sea VERDADERA, y sólo producirá FALSO como resultado cuando todas las expresiones lógicas de entrada sean falsas.
Ejemplos: OR (5 = 3 + 3, 2 = 1 + 1) = = VERDADERO
OR (3 + 2 + 6, 2 = 1 + 2) = = FALSO



La función IF es similar a la instrucción IF propia del lenguaje BASIC: si se cumple la condición C, el resultado es T; por el contrario, si no se cumple la condición C, el resultado será E.

5. NOT (L)

El único argumento de esta función debe ser una expresión lógica. El resultado obtenido al aplicar NOT a L será VERDADERO si L es FALSO, y viceversa: será FALSO si L es VERDADERO. Por ejemplo: NOT (3 = 2 + 1) = FALSO, NOT (3 = 2 + 2) = VERDADERO.

6. IF (L,N,M)

Los argumentos de la función IF son mixtos: el primero, L, debe consistir en una expresión lógica; en cambio, los dos siguientes, N y M, pueden ser valores de cualquier tipo. El resultado producido por la función IF será N en el caso de que la expresión L sea VERDADERA, y será M cuando L sea FALSA. Ejemplos:
IF (5 = 2 + 3, "BIEN", "MAL") = "BIEN",
IF (5 = 3 + 3, "BIEN", "MAL") = "MAL"

7. ISNA (N)

La función ISNA sirve para detectar el uso de datos no disponibles. Para ello, cuando su único argumento no esté disponible, el resultado será VERDADERO; en cambio, si el argumento está disponible, su resultado será FALSO. Generalmente, su utilización va ligada a la función IF. Por ejemplo: IF (ISNA(R3C4), o, R3C4), con esta combinación de funciones se obtendría 0 si la celda de la hoja R3C4 no estuviese disponible; en el caso de que sí lo estuviese, el resultado sería, precisamente, el contenido de dicha celda.

8. ISERROR (N)

Esta función es una generalización de la anterior extendida a cualquier error; esto es: devuelve el valor lógico VERDADERO si se produce un error, y el valor FALSO si no se produce.

9. NA ()

Mediante la función NA se puede almacenar el valor especial "no disponible" para que la función ISNA pueda detectar la no disponibilidad.

FUNCIONES FINANCIERAS

1. AVERAGE (lista de variables)

Esta función calcula el valor medio de la lista de variables numéricas que figuran en su argumento. Por ejemplo: AVERAGE (3,5,7) = 5.

2. SUM (lista de variables)

La función sumatorio, SUM, permite calcuar el valor correspondiente a la suma de los parámetros de entrada. Por ejemplo: SUM (3,5,7) = 15.

3. COUNT (lista de variables)

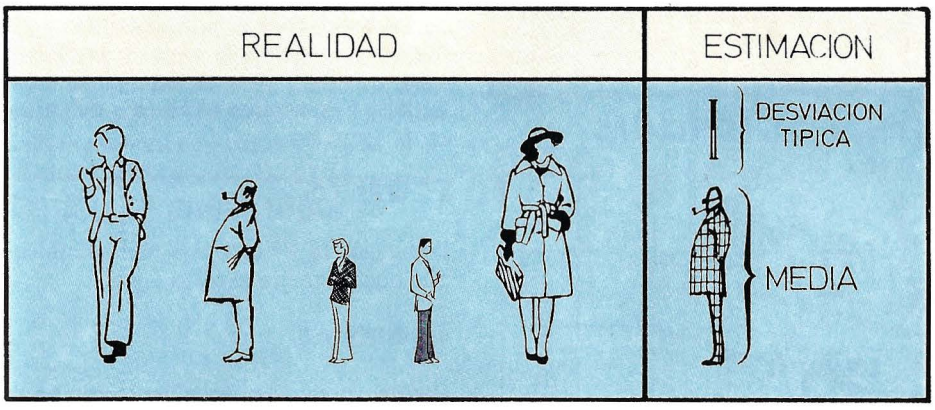
Mediante esta función se obtiene como resultado el número natural que coincide con la cantidad de valores que forman la lista de variables: COUNT (3, 5, 7) = 3.

4. STDEV (lista de variables)

Calcula la desviación típica existente entre la lista de variables de entrada. Para ello, si suponemos que la lista de variables está formada por $X_1, X_2, \ldots, X_n$, la desviación típica se calcula mediante la siguiente fórmula:

$$\text{STDEV}(X_1 \ldots X_n) = \sqrt{\frac{\sum_{i=1}^{n} X_i^2 - \frac{\left(\sum_{i=1}^{n} X_i\right)^2}{n}}{n-1}}$$

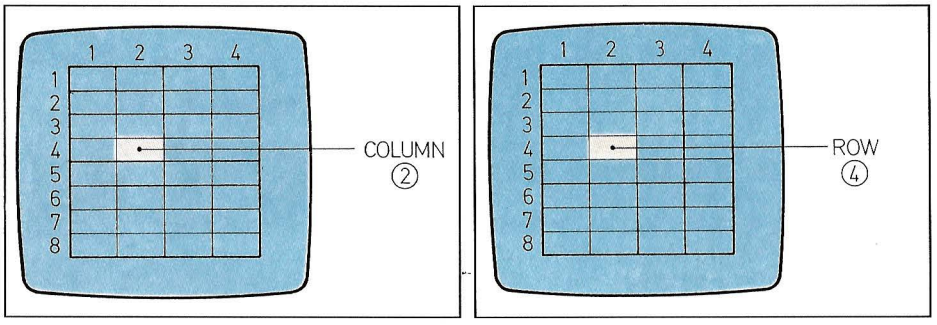
Por ejemplo: STDEV (3,5,7) = 2



La media consiste en una estimación centrada de la realidad, mientras que la desviación típica informa sobre las diferencias existentes entre la realidad y la media.

FUNCIONES DEL MULTIPLAN

Funciones matemáticas	Funciones lógicas	Funciones financieras	Funciones especiales
ABS	TRUE	AVERAGE	COLUMN
SIGN	FALSE	SUM	ROW
SIN	AND	COUNT	DOLLAR
COS	OR	STEDEV	FIXED
TAN	NOT	MAX	VALUE
ATAN	IF	MIN	LEN
EXP	ISNA	NPV	MID
LN	ISERROR	LOOKUP	REPT
LOG10	NA		INDEX
INT			
ROUND			
SQRT			
MOD			



La función COLUMN entrega como resultado un número coincidente con la columna de la celda en la que se incluye dicha función. Análogamente, la función ROW devuelve el número de la respectiva fila.

5. MAX (lista de variables)

La función máximo MAX se utiliza para obtener el valor más alto de todos los argumentos de entrada. Por ejemplo: MAX (3, 5, 7) = 7.

6. MIN (lista de variables)

Su misión es obtener el valor más pequeño de los números que forman la lista de variables. Por ejemplo: MIN (3,5,7) = 3.

7. NPV (N, lista de variables)

Calcula el valor neto actual del "cash flow" de la lista de variables, con un porcentaje de descuento P. Para ello utiliza la siguiente fórmula:

$$NPV\ (N, X_1, \ldots, X_n) = \sum_{i=1}^{n} \frac{X_i}{(1+N)^i}$$

Por ejemplo: NPV (0.1, 15, 17, 19) = 41.96

8. LOOKUP (N, Tabla)

La función LOOKUP busca N en la primera fila o columna de la tabla y devuelve el valor de una celda de la última fila o columna de la tabla.

FUNCIONES ESPECIALES

1. COLUMN ()

Esta función no necesita ningún argumento y devuelve el número de la columna en la que se encuentra la fórmula donde es invocada. Por ejemplo, si en la columna 3 aparece la fórmula 1980 + COLUMN (), el resultado será 1983.

2. ROW ()

Análogamente, la función ROW produce como resultado la fila en que se encuentra la fórmula que la contiene. Por ejemplo si en la celda R5C2 incluimos la fórmula ROW() * 10 + COLUMN(), el resultado producido será 52.

3. DOLLAR (N)

Convierte el valor que obtiene como argumento al formato dolar; es decir, inserta el carácter "$" a la izquierda del número. Por ejemplo: DOLLAR (527) = $ 527.00.

4. FIXED (N,M)

La función FIXED necesita como argumentos dos valores, N y M, que aportan un valor numérico y un número de dígitos decimales.

5. VALUE (T)

Esta función se utiliza para recuperar la información de la celda activa y para ello utiliza el parámetro T como salida.

6. LEN (T)

La función LEN devuelve como resultado el número de caracteres que contiene el texto.

7. MID (T,S,C)

Sirve para tomar parte de la información de un texto, para ello necesita tres argumentos: T contendrá el texto de selección, S la primera posición que se debe tomar y C especifica el número de caracteres tomados.

8. REPT (T,C)

La función REPT cuenta el número de veces que se repite el texto T en C.

9. INDEX (A,S)

Permite obtener el valor contenido en el área A a partir de los subíndices S en que se encuentre la información requerida.

Funciones a medida

Creación y uso de funciones definidas por el usuario

A la hora de construir un programa hay que echar mano de las palabras clave del BASIC. Estas, como se sabe, pueden ser comandos o funciones. Las funciones son aquellas palabras reservadas cuya ejecución proporciona un dato. Un ejemplo de función es SIN, cuyo cometido es calcular el valor del seno de un determinado dato numérico (valor del ángulo expresado en radianes). Un ejemplo de su modo de actuación es el siguiente:

```
PRINT SIN(3)
 .14112
■
```

Normalmente, las funciones hacen uso de un dato, del que extraen la información para calcular el resultado pedido. En el ejemplo precedente, el dato es el número 3. Estos datos que son utilizados por las funciones reciben el apelativo de "datos de entrada". De la misma forma, los datos calculados por la función (resultados) reciben el nombre de "datos de salida".

EL USO DE SUBRUTINAS

Los intérpretes del lenguaje BASIC suelen incluir un nutrido conjunto de funciones. Entre ellas se encuentran las ya estudiadas para el manejo de datos numéricos o alfanuméricos.

En ocasiones, puede resultar útil crear funciones específicas, no incluidas directamente en el traductor BASIC, para determinados cometidos. Por ejemplo, si se desea calcular reiteradamente el 10 % de una serie de números, no es práctico repetir varias veces el mismo bloque de instrucciones. Para realizar dicho cálculo sobre un dato contenido en la variable X será necesaria la siguiente línea BASIC.

```
10 LET DPC=(X/100)*10
```

Al ejecutarla, el resultado se almacena en la variable DPC. Para evitar la continua repetición del contenido de esta línea se puede hacer uso de una subrutina. Una subrutina, como ya se comentó en un capítulo anterior, es un fragmento de programa que puede ser utilizado varias veces por el programa principal. La subrutina encargada de calcular el 10 % tomaría este aspecto:

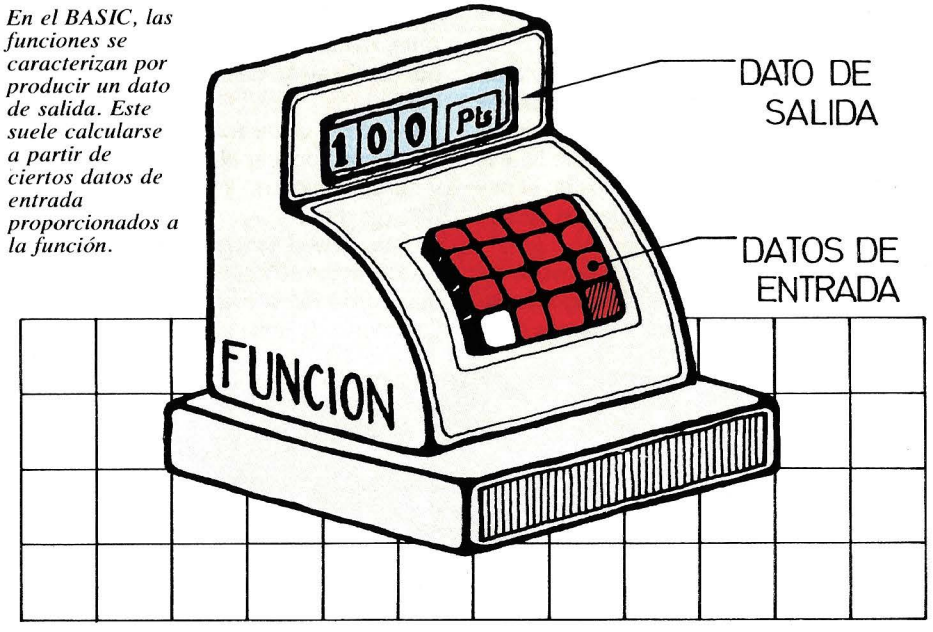
```
1000 LET DPC=(X/100)*10
1010 RETURN
```

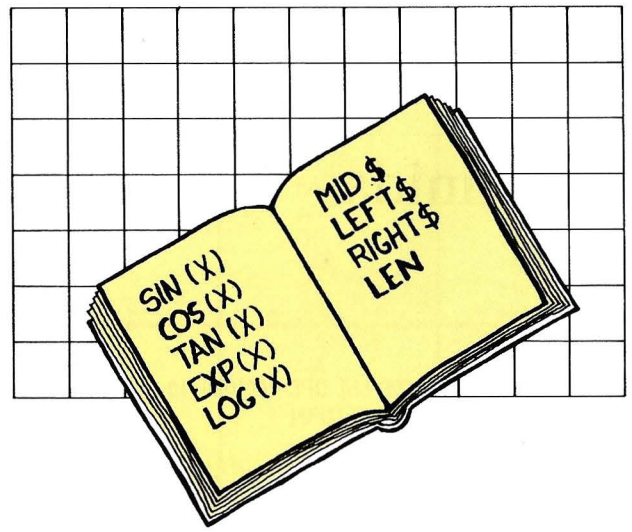
En este caso, el dato de entrada de la "función 10 %" se debe almacenar en la variable X. Una vez que se ha ejecutado la subrutina, el resultado aparecerá en la variable DPC. El uso de esta pseudofunción implicará tres pasos. El primero será situar el número base (del que se desea extraer el 10 %) en la variable X. Después se ha de "llamar" a la subrutina; como quiera que ésta se encuentra en la línea 1000, se utilizará una instrucción GOSUB 1000. El último paso consiste en rescatar el resultado de la variable DPC. Estos tres pasos se plasmarían en BASIC de la siguiente forma:

```
10 LET X=15000
20 GOSUB 1000
30 PRINT DPC
```

Los referidos pasos habrán de ejecutarse

En el BASIC, las funciones se caracterizan por producir un dato de salida. Este suele calcularse a partir de ciertos datos de entrada proporcionados a la función.





La mayor parte de los dialectos BASIC incorporan una nutrida colección de funciones predefinidas.

cada vez que se quiera hacer uso de la subrutina.
Como ejemplo más completo, vamos a examinar un programa capaz de restar el 10 % a una lista de números. Los datos iniciales han de residir en un "array" o variable suscrita denominada, por ejemplo, BRUTO. Los resultados se almacenarán en otro array, que en este caso llevará por nombre NETO. En definitiva, el núcleo del programa estará constituido por las siguientes líneas:

```
100 FOR I=1 TO 100
110 LET X=BRUTO(I)
120 GOSUB 1000
130 LET NETO(I)=BRUTO(I)-DPC
140 NEXT I
```

En el listado se observa que el resultado se utiliza directamente en la sentencia de asignación de la línea 130. Si la función a crear necesita más datos de entrada, éstos tendrán que ser actualizados en el primer paso, previo a la llamada a la subrutina.
Las funciones construidas con la ayuda de subrutinas son ligeramente engorrosas. Lo ideal sería disponer de funciones definibles que adoptaran el aspecto propio de las funciones que el BASIC ofrece dentro de su repertorio.
Veamos cuáles son las diferencias entre la función definida previamente y una función existente en el seno del intérprete; por ejemplo, la función seno.
Para hacer uso de la función seno basta con recurrir a la palabra clave SIN, mientras que la función definitiva necesita de un GOSUB 1000. El dato de entrada de SIN se adjunta como argumento de la función; es decir, acompaña a dicha palabra encerrado entre paréntesis. En el caso del tanto por ciento, es preciso realizar una asignación previa de la variable adecuada (X). Además, el resultado de la función SIN se obtiene automáticamente al utilizarla, mientras que el resultado del tanto por ciento se ha de extraer de una variable (DPC).
En el caso de la función implementada, basta con conocer el nombre de la misma y su utilización no implica la inclusión de nuevas líneas de programa; sencillamente, puede emplearse en una sentencia de asignación o como argumento de otro comando o función.
La función definida por medio de una subrutina necesita para su uso de más información. Es preciso conocer su localización: el número de línea donde se encuentra la subrutina. Además de ello, es imprescindible tener conocimiento de los nombres de las variables que utiliza (en el ejemplo X y DPC). Este método implica un cierto espacio en el listado; ni más ni menos, el dedicado a la consecución de los tres pasos arriba mencionados.
En BASIC existe otro método para la definición de funciones: se trata de una facilidad que incorpora el propio intérprete bajo los auspicios del comando DEF FN.

DEFINICION DE FUNCIONES

El comando DEF FN permite al usuario definir sus propias funciones: éstas servirán para aliviar la confección del resto del programa. Su ámbito de actuación se limita al programa en el que se han definido. Esto es, si se introduce un nuevo programa no se conservarán las antiguas funciones. La definición de una función de usuario se realiza en una sola línea de programa. En ella se deben incluir las palabras clave DEF y FN, contiguas y por ese orden; tras ellas, se escribe la definición de la función.
El formato de la definición de funciones muestra el siguiente aspecto:

(número de línea) DEF FN<nombre>
[(<variable>)]=<operaciones>

En dicha expresión cabe distinguir tres partes fundamentales, acompañando a las palabras DEF FN. Junto a la partícula FN se coloca un dato alfanumérico que servirá de nombre a la función. A continuación, y entre paréntesis, se sitúa un nombre de variable que se utilizará como dato de entrada de la función. Por último, y tras el signo "igual", se formulan las operaciones que calcularán el dato de salida.
Un ejemplo de función definida es el siguiente:

```
10 DEF FND(X)=(X/100)*10
```

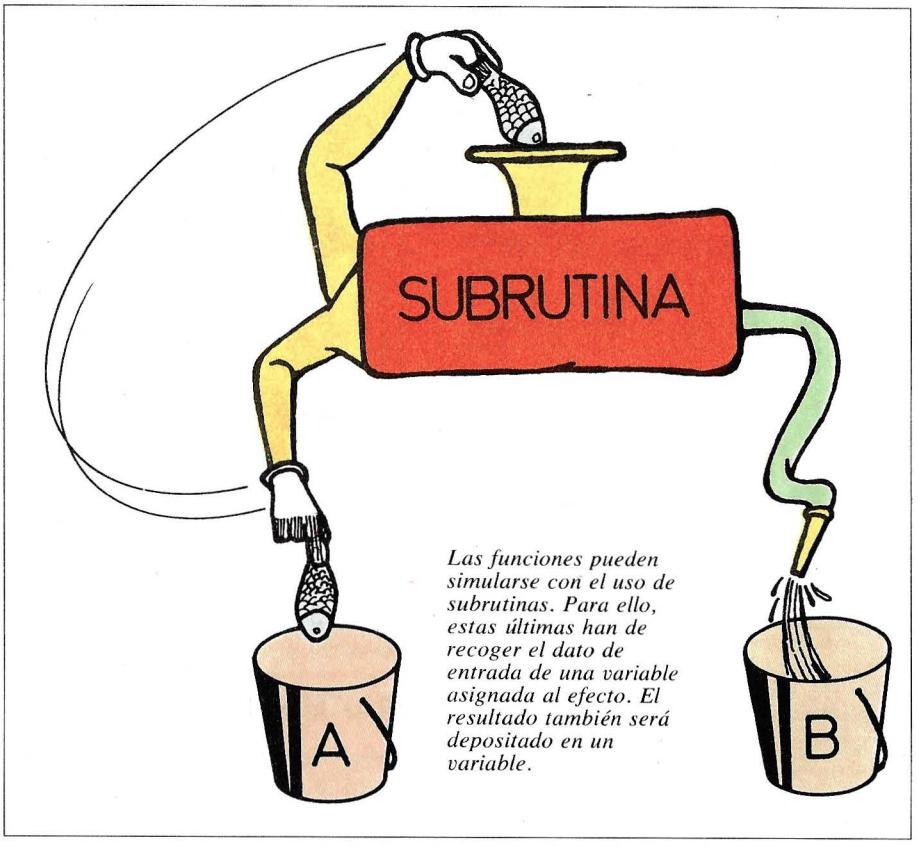
Se ha utilizado el mismo ejemplo que al principio del capítulo. En esta ocasión la función recibe el nombre "D". Ese es, precisamente, el carácter que acompaña a FN.
La variable de referencia, utilizada en el

DEF FN

Define la función de usuario indicada por las instrucciones situadas en su argumento.

Formato: DEF FN<nombre>[(<parámetros>)]=<expresión>
Ejemplos: 10 DEF FNA(X)=SIN(X)+X^2
50 DEF FNPI=3.141592

cuerpo de la definición, es X. El nombre de variable, encerrado entre paréntesis, puede ser cualquiera. La única restricción está en que dicho nombre de variable debe coincidir con el que se emplee como dato de entrada en la definición. En el caso enunciado, X sirve para calcular el 10 % de ese valor.
La variable referencial es una variable "ficticia". En efecto, no se trata estrictamente de una variable auténtica, sino que sólo sirve para indicar cuál de los datos de la definición es el de entrada. En el interior de la definición se pueden incluir más variables, cuyos valores serán aquellos que tengan asignados en el momento de hacer uso de la función.



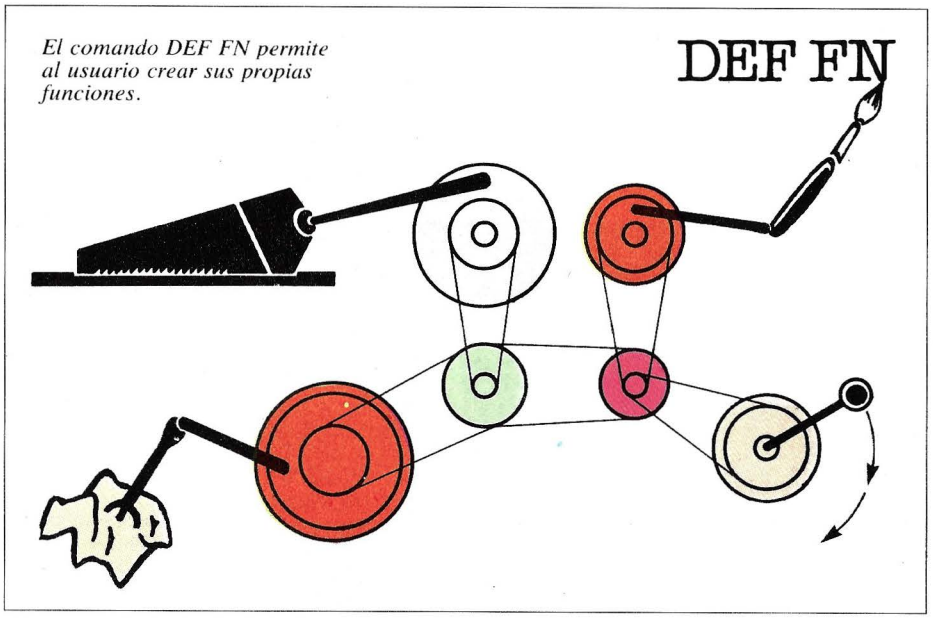
Las funciones pueden simularse con el uso de subrutinas. Para ello, estas últimas han de recoger el dato de entrada de una variable asignada al efecto. El resultado también será depositado en un variable.

USO Y DISFRUTE DE LAS FUNCIONES DE USUARIO

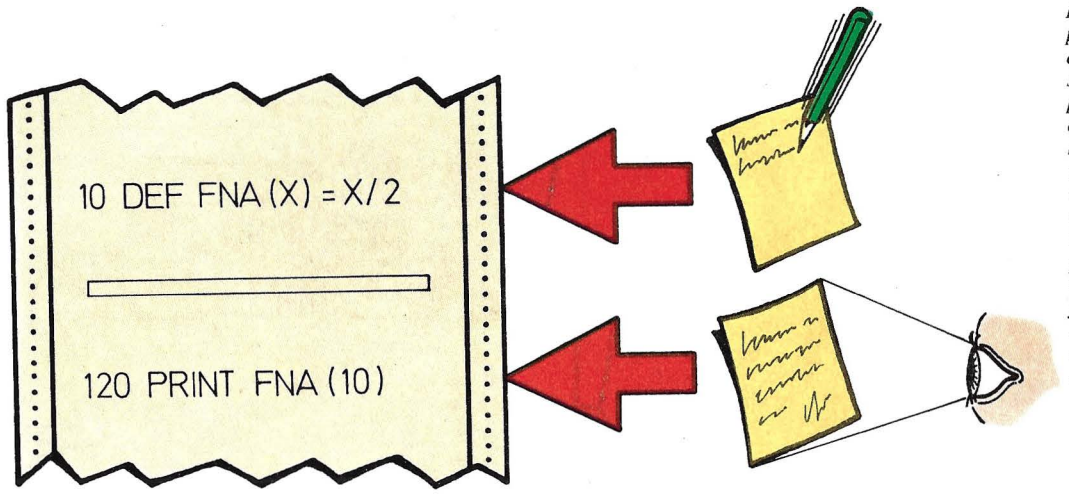
Se ha estudiado ya el modo de definir una función de usuario. Este es el paso previo, pero sin duda la faceta más importante es la de su puesta en práctica.
Para acceder a una función definida hay que actuar de la misma forma que para acceder a una función propia del intérprete BASIC. En principio, se escribirá el nombre de la función (nombre otorgado en la definición de la misma); no obstante, para indicar que se trata de una función definida por el usuario, su nombre debe ir precedido de las letras FN. Esta sería la forma de utilizar la función definida en el apartado precedente:

```
PRINT FND(50)<CR>
5
■
```

En el ejemplo se pone de manifiesto su semejanza respecto al uso de las funciones propias del intérprete BASIC. Al igual que con aquellas, el argumento indica el dato de entrada. Dicho argumento se sitúa entre paréntesis.
Para que sea posible utilizar una de estas funciones, es preciso definirla previamente. Ello implica dos cosas. Por una parte, ha de existir en el seno del programa una línea que defina la función por medio del comando DEF FN. Otro requisito es que el ordenador conozca dicha definición. Para ello la ejecución ha de pasar por la susodicha línea de definición, puesto que si no se ejecuta esa línea, la función no será utilizable, mostrando el aparato un mensaje de error.



El comando DEF FN permite al usuario crear sus propias funciones.



Para la puesta en práctica de las funciones definidas por el usuario, son necesarios dos pasos. En el primero, el ordenador toma nota de la función definida por medio del comando DEF FN. Posteriormente, cuando ésta sea utilizada dentro del programa, la máquina revisará la lista de funciones definidas para aplicarla y calcular el correspondiente resultado.

El uso de funciones definidas da una mayor flexibilidad a la programación. Como ejemplo comparativo vamos a construir la rutina mencionada al principio, utilizando ahora una función. Este sería el listado de la misma.

```
10  DEF FND(X)=(X/100)*10
100 FOR I=1 TO 100
130 LET NETO(I)=BRUTO(I)-FND(BRUTO(I))
140 NEXT I
```

Al emplear la función se han ahorrado dos líneas de programa. Además, las líneas de la subrutina se ven sustituidas por la de definición, que resulta más corta.
Una función definida puede utilizarse en cualquier ocasión que lo requiera: puede incluirse en una sentencia de asignación, o incluso en la definición de otra función. Dentro del ejemplo que nos ocupa, podría definirse una nueva función.

```
20 DEF FNN(X)=X-FND(X)
```

Con esta función, la rutina del ejemplo se simplifica aun en mayor medida. La función FNN procede a calcular el valor que resulta de restar el 10 % a una cantidad. Dicho 10 % se calcula con la función FND. De esta forma, el ejemplo se reduciría al siguiente bloque de instrucciones:

```
10  DEF FND(X)=(X/100)*10
20  DEF FNN(X)=X-FND(X)
100 FOR I=1 TO 100
130 LET NETO(I)=FNN(BRUTO(I))
140 NEXT I
```

BIBLIOTECA DE FUNCIONES

Para redundar en el aspecto práctico, se analizan a continuación algunos ejemplos de definición de funciones. En ellos se encuentran funciones no habituales en los intérpretes BASIC, así como otras de uso frecuente. Cada grupo de funciones puede resultar útil en distintos tipos de programa. Y todas ellas formarán un "cajón de sastre" o biblioteca de funciones diversas que convendrá tener a mano a la hora de ponerse a programar.
La mayor utilidad de las funciones reside en el cálculo de fórmulas matemáticas. Este es un hecho que vamos a considerar en nuestro primer grupo de funciones definidas, cuyo ámbito será la trigonometría.
La mayor parte de los traductores BASIC disponen de cuatro funciones trigonométricas: seno (SIN), coseno (COS), tangente (TAN) y arco tangente (ATN). Remitiéndonos a un capítulo anterior, vamos a obtener ahora las restantes funciones trigonométricas; las primeras serán la secante, cosecante y cotangente.

```
10 DEF FNSEC(X)=1/COS(X)
20 DEF FNCSC(X)=1/SIN(X)
30 DEF FNCTG(X)=1/TAN(X)
```

La simplicidad de estas tres funciones obvia su explicación. Algo más complicadas resultan las funciones arco seno y arco coseno. Las fórmulas que las relacionan con ATN han sido ya comentadas en un capítulo anterior de la obra; éstas son:

```
ASN(X)=ATN(X/SQR(1-X^2))
ACS(X)=ATN(SQR(1-X^2)/X)
```

Su definición como funciones directas no tiene mayor complicación que la de copiar sus fórmulas en un comando DEF FN. Aquí está el resultado:

```
40 DEF FNAS(X)=ATN(X/SQR(1-X^2))
50 DEF FNAC(X)=ATN(SQR(1-X^2)/X)
```

Otras funciones útiles en numerosas ocasiones son las hiperbólicas. El seno y coseno hiperbólico responden a las siguientes fórmulas:

```
SH(X)=(EXP(X)-EXP(-X))/2
CH(X)=(EXP(X)+EXP(-X))/2
```

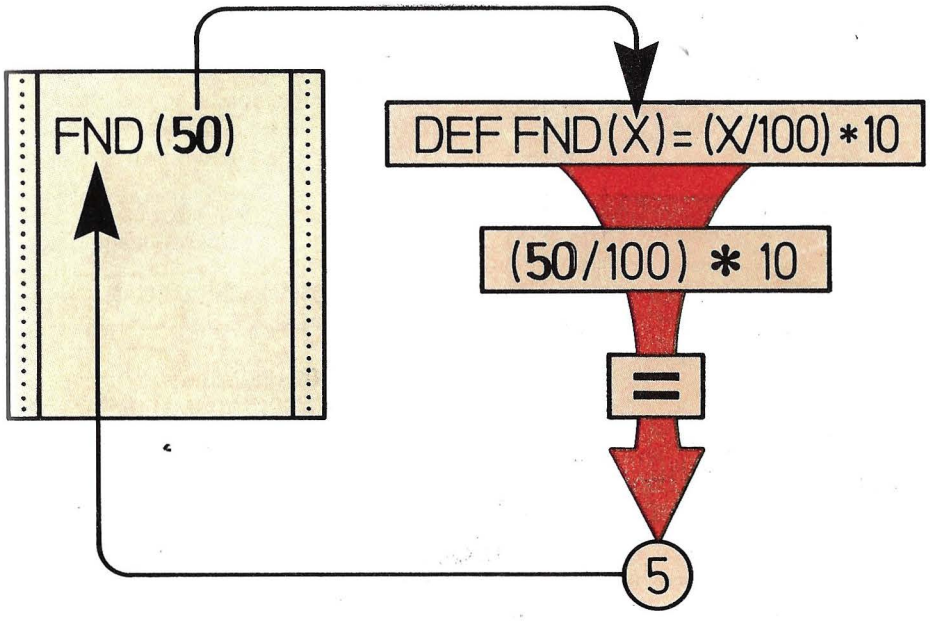
Según estas expresiones, se pueden crear muy fácilmente las funciones que los calculan:

```
60 DEF FNSH(X)=(EXP(X)-EXP(-X))/2
70 DEF FNCH(X)=(EXP(X)+EXP(-X))/2
```

Para definir la tangente hiperbólica se hará uso del seno y coseno. La tangente es igual al cociente entre seno y coseno, con lo que su definición se puede reducir a una llamada a las funciones anteriores.

```
80 DEF FNTH(X)=FNSH(X)/FNCH(X)
```

Siguiendo con las funciones poco habitua-



Al utilizar una función definida, el ordenador evalúa la expresión contenida en su definición. Para ello, sustituye la variable de entrada por el valor incluido como argumento de la función.

les vamos a entrar ahora en el campo de los logaritmos. La primera función a definir es la que obtendrá directamente el logaritmo decimal. La fórmula a aplicar es la siguiente.

```
LOG 10(X)=LOG(X)/LOG(10)
```

En ella se hace uso de la función logaritmo neperiano (LOG) presente en la práctica totalidad de ordenadores. Su definición se reduce, pues, a la siguiente línea de programa:

```
90 DEF FNLOG10(X)=LOG(X)=LOG(10)
```

Una vez que se dispone de la función logaritmo decimal, nada resulta más fácil que sintetizar su función inversa: el antilogaritmo. El cálculo del antilogaritmo decimal se resume a elevar la base 10 al dato del que se desea conocer su antilogaritmo. Por lo tanto, la función que proporcione ese resultado será bien fácil de definir.

```
100 DEF FNAL10(X)=10^X
```

Como punto final a esta colección de funciones, se ofrece el listado correspondiente a la definición a las diez nuevas herramientas de cálculo:

```
10 DEF FNSEC(X)=1/COS(X)
20 DEF FNCSC(X)=1/SIN(X)
30 DEF FNCTG(X)=1/TAN(X)
40 DEF FNAS(X)=ATN(X/SQR(1-X^2))
50 DEF FNAC(X)=ATN(SQR(1-X^2)/X)
60 DEF FNSH(X)=(EXP(X)-EXP(-X))/2
70 DEF FNCH(X)=(EXP(X)+EXP(-X))/2
80 DEF FNTH(X)=FNSH(X)/FNCH(X)
90 DEF FNLOG10(X)=LOG(X)/LOG(10)
100 DEF FNAL10(X)=10^X
```

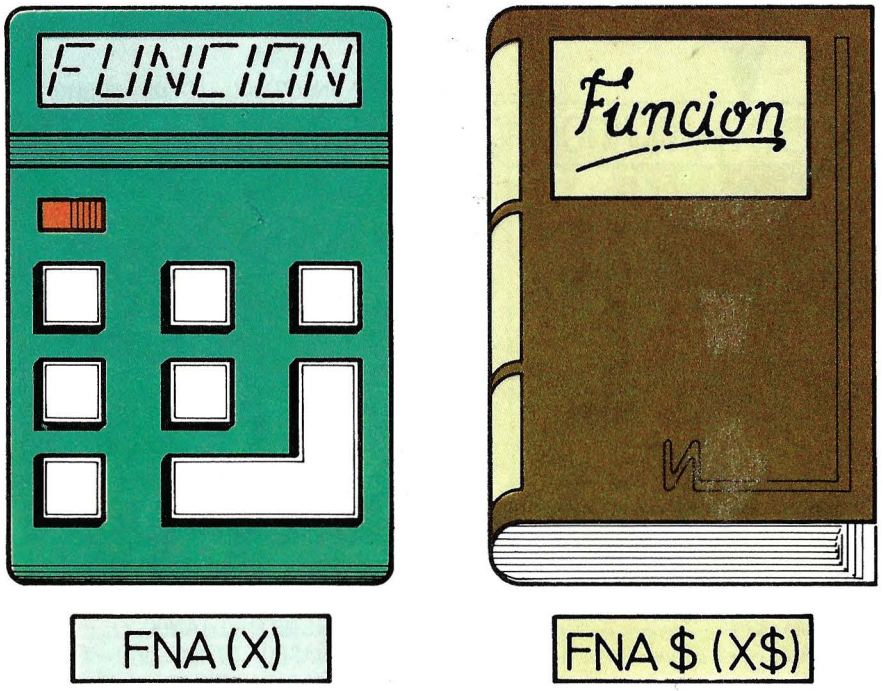
DEF FN en el QL

Algunos dialectos BASIC se alejan de la norma en ciertos aspectos. Ello puede contribuir a una programación más potente, aunque también puede resultar más compleja. Este es el caso del BASIC del QL. El intérprete BASIC del QL (el denominado SUPERBASIC) permite una gran flexibilidad en la definición de funciones. El primer lugar, éstas no se ven limitadas a una línea de programa. Una definición puede ocupar varias líneas, siempre y cuando ello se indique. Las referidas líneas deben estar encabezadas por una en la que aparezcan las palabras reservadas DEF FN. Junto a estas dos palabras se sitúan el nombre y los habituales parámetros de la función. Las líneas siguientes son las que definen el cometido de la función. Una vez completada la definición, ésta se cierra con las palabras END DEF.
En el interior de la función es posible incluir todo tipo de comandos y funciones, incluso bucles de tipo FOR/NEXT; y también se pueden definir variables cuyo ámbito de utilización se limite a la propia función. Estas variables se denominan locales y se identifican con el prefijo LOC.

De los cálculos que se efectúen por medio de una función sólo se devolverá un dato como resultado. Para indicar el valor de salida que debe proporcionar la función se utiliza la partícula RET.
Las funciones definidas en este dialecto gozan de todas las propiedades generales: múltiples parámetros, manejo de cadenas, etc.
Como ejemplo de definición de funciones en este curioso dialecto, se incluye una función que calcula el factorial de un número (n!=1*2*...*n).

```
10DEF FN factorial (n)
20 LOC suma
30 LET suma=1
40 FOR I=1 TO n
50 suma=suma*I
60 END FOR n (equivalente a NEXT n.)
70 RET suma
80END DEF
```

Al igual que en otros muchos aspectos, la definición de funciones con el Super-BASIC del Sinclair QL muestra claras diferencias con respecto al procedimiento habitual.



Habitualmente, los intérpretes BASIC permiten la definición de funciones adecuadas para trabajar tanto con datos numéricos como alfanuméricos. Estas últimas deben unir a su nombre el símbolo $.

FUNCIONES NO MATEMATICAS

La definición de funciones no se limita a los cálculos matemáticos. También es posible la creación de funciones que traten con otros tipos de datos. Por ejemplo, se puede definir una función que extraiga la inicial de una cadena de caracteres. Este sería su aspecto:

```
10 DEF FNINI$(A$)=LEFT$(A$,1)
```

El uso de esta nueva función es similar al de cualquier otra función, sea o no de tipo matemático. La única salvedad aparece en que ahora se utilizan datos de cadena de caracteres. Esta circunstancia la revela la presencia del signo de "dólar" al final del nombre de la variable. Como el resultado ha de ser un nuevo dato alfanumérico el nombre de la función ha de llevar también ese identificador de tipo.
En ciertos casos pueden incluso mezclarse datos de distinto tipo dentro de una misma función. Esto ha de hacerse observando la precaución de no aplicar funciones numéricas a datos no numéricos, y viceversa. Veamos un ejemplo.

La función que sigue sirve para codificar mensajes.

```
10 DEF FNCO$(A$)=CHR$(ASC(A$)+2)
```

Esta admite un dato alfanumérico consistente en una letra, de la que extrae su correspondiente código ASCII (por medio de ASC(A$)). A ese dato numérico le suma el valor 2, alterando su código en consecuencia, y entrega como dato de salida el carácter asociado al nuevo código. En definitiva, la función devuelve una letra relacionada con la propuesta como dato de entrada. Ello puede servir, por ejemplo, para codificar mensajes secretos.
El siguiente programa hace uso de la referida función para codificar una palabra:

```
10 DEF FNCO$(A$)=CHR$(ASC(A$)+2)
20 INPUT "MENSAJE";M$
30 LET C$=""
40 FOR I=1 TO LEN(M$)
50 C$=C$+FNCO$(MID$(M$,I,1))
60 NEXT I
70 PRINT C$
```

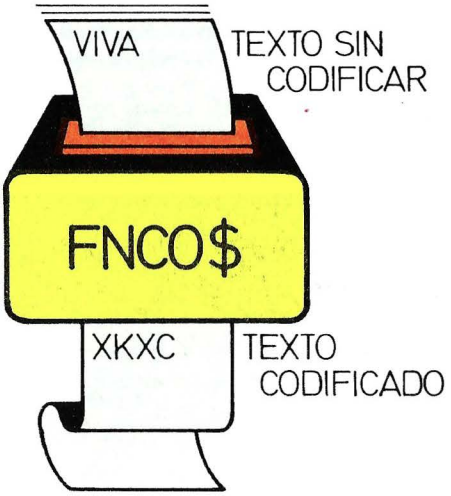
El mensaje se desglosa en los correspondientes caracteres; acción ésta que realiza la función MID$(M$,I,1) de la línea 50. A cada carácter se le aplica la función predefinida. Proceso que se repite hasta convertir todos los caracteres del mensaje inicial (FOR I=1 TO LEN(M$)). Los sucesivos caracteres codificados se van almacenando en la variable alfanumérica C$. Al final, se muestra el resultado obtenido en dicha variable.
La ejecución de este pequeño programa dará el siguiente resultado:

```
RUN<CR>
MENSAJE? PEPE
RGRG
■
```

En la ejecución se observa como el mensaje PEPE se transforma en RGRG. Esta clave (u otra parecida) puede ser útil para intercambiar mensajes secretos.
El programa adecuado para decodificar los mensajes en clave coincidirá con el anterior; la única diferencia aparecerá en la función predefinida. La función de decodificación será la siguiente.

```
10 DEF FNDE$(A$)=CHR$(ASC(A$)-2)
```

En ella se realiza la acción contraria, restando dos unidades a cada código para restablecer el código real.



La función FNCO$, definida en el texto, sirve para codificar mensajes por medio de una clave secreta.

TABLA DE CONVERSION

ORDENADOR	DEF FN		
	DEF FN<n>(<p>)=<exp. num.>	DEF FN<n>(<p>)=<exp. cad.>	DEF FN<n>=<exp.>
APPLE II (APPLESOFT)	—	—	—
APRICOT	DEF FN<n>(<p>)=<exp. num.>	DEF FN<n>(<p>)=<exp. cad.>	DEF FN<n>=<exp.>
ATARI	—	—	—
CBM64	DEF FN<n>(<p>)=<exp. num.>	—	—
DRAGON	DEF FN<n>(<p>)=<exp. num.>	—	—
EQUIPOS MSX	DEF FN<n>(<p>)=<exp. num.>	DEF FN<n>(<p>)=<exp. cad.>	DEF FN<n>=<exp.>
HP-150	DEF FN<n>(<p>)=<exp. num.>	DEF FN<n>(<p>)=<exp. cad.>	DEF FN<n>=<exp.>
IBM PC	DEF FN<n>(<p>)=<exp. num.>	DEF FN<n>(<p>)=<exp. cad.>	DEF FN<n>=<exp.>
MPF	DEF FN<n>(<p>)=<exp. num.>	—	—
NCR DM-V (MS)-BASIC)	DEF FN<n>(<p>)=<exp. num.>	—	DEF FN<n>=<exp.>
NEW BRAIN	DEF FN<n>(<p>)=<exp. num.>	DEF FN<n>(<p>)=<exp. cad.>	DEF FN<n>=<exp.>
ORIC	DEF FN<n>(<p>)=<exp. num.>	—	—
SHARP MZ-700 (MZ-BASIC)	DEF FN<n>(<p>)=<exp. num.>	—	—
SINCLAIR QL	*	*	*
SPECTRAVIDEO	DEF FN<n>(<p>)=<exp. num.>	DEF FN<n>(<p>)=<exp. cad.>	DEF FN<n>=<exp.>
ZX-SPECTRUM	DEF FN<n>(<p>)=<exp. num.>	DEF FN<n>(<p>)=<exp. cad.>	DEF FN<n>=<exp.>

<n>: nombre de la función. <p>: parámetro. <exp. num.>: expresión numérica. <exp. cad.>: expresión alfanumérica. DEF FN<n>=<exp.>: definición con número de parámetros variable. *: el QL permite definiciones en más de una línea.

FUNCIONES SIN PARAMETROS

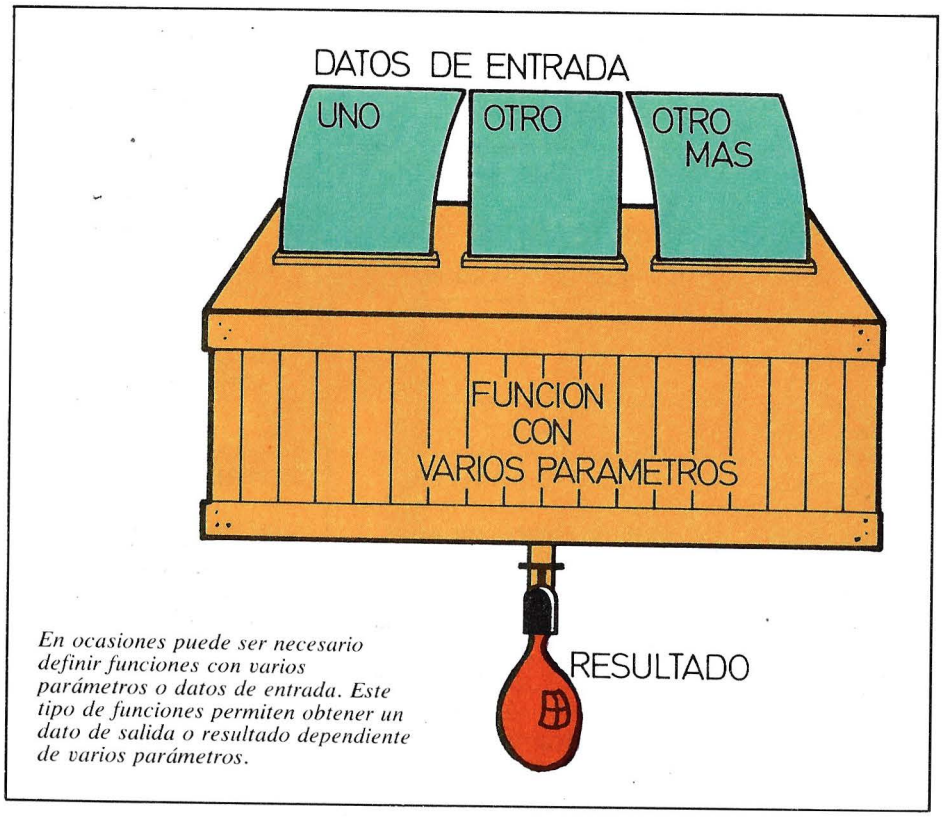
En determinadas ocasiones se utilizan funciones en las que el dato de salida es fijo. Por ejemplo, en los equipos que no disponen de un comando para el borrado de pantalla, éste se puede generar por medio de una función:

```
10 DEF FNC$(X)=CHR$(26)
20 PRINT FNC$(0)
```

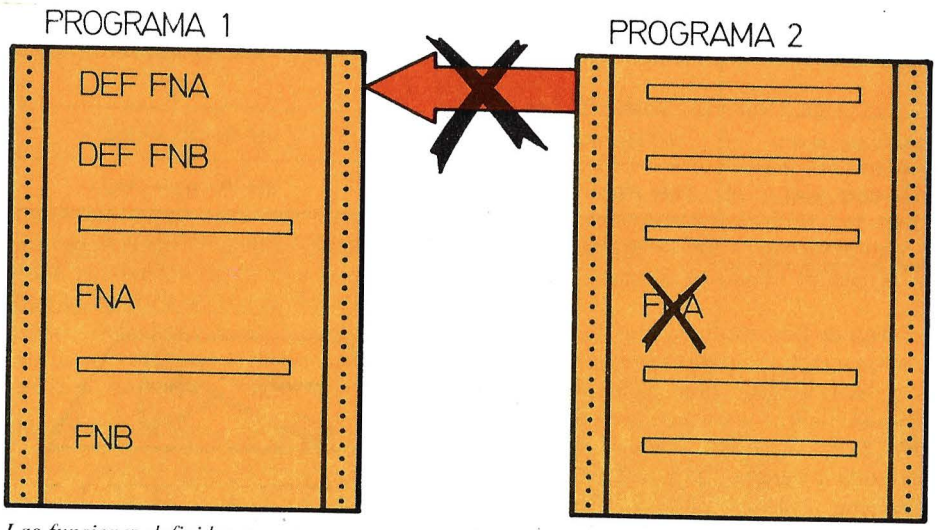
Esta sería la función adecuada para aquellos casos en los que el borrado de pantalla está asociado al código ASCII 26. Si el código de borrado fuera otro, bastaría con cambiar dicho número.

Con esta función definida el borrado se logrará sin más que teclear PRINT FNC$(0). En nuestro ejemplo, el dato de entrada o parámetro de la función no interviene en absoluto; de ahí que no importe el dato que se incluya, puesto que éste no tiene repercusión alguna en la función. Realmente, cuando no se necesita parámetro de entrada, éste puede obviarse en la definición. Así pues, la función anterior se puede reducir en su expresión:

Las funciones sin parámetros proporcionan un dato fijo. En consecuencia, su resultado será el mismo cada vez que ésta sea utilizada dentro del programa.



En ocasiones puede ser necesario definir funciones con varios parámetros o datos de entrada. Este tipo de funciones permiten obtener un dato de salida o resultado dependiente de varios parámetros.



Las funciones definidas en un programa no pueden ser compartidas por otros programas. En consecuencia, para hacer uso de una función de esta índole, es obligado definirla dentro del mismo programa en el que vaya a ser utilizada.

```
10 DEF FNC$=CHR$(26)
20 PRINT FNC$
```

Ahora, el borrado de pantalla resultará aún más cómodo: basta con teclear PRINT FNC$.

Las funciones sin parámetros pueden utilizarse, por ejemplo, para definir las constantes π (número PI) o *e* (número "e").

Este es precisamente el cometido de las dos definiciones que siguen:

```
20 DEF FNPI=3.141592
30 DEF FNE=EXP(1)
```

Haciendo uso de la primera, el cálculo de la longitud de la circunferencia se resume a la siguiente función:

```
100 DEF FNL(R)=2*FNPI*R
```

¡MAS PARAMETROS!

Una función puede prescindir de parámetros, como ya se ha señalado. No obstante, también admitir la situación opuesta: utilizar más de un parámetro. En tal caso, éstos se separan por medio de comas, dentro de la zona encerrada por paréntesis.

Las funciones con múltiples parámetros resultan útiles para programar fórmulas. Por ejemplo, para el cálculo de la velocidad media de un móvil partiendo del espacio recorrido y el tiempo invertido:

```
100 DEF FNV(E,T)=E/T
```

Desde luego, pueden crearse funciones bastante más complejas. La que sigue calcula las calorías producidas al pasar una corriente eléctrica por un conductor (Ley de Joule).

```
200 DEF FNCA(R,I,T)=.24*R*I^2*T
```

En los parámetros se especifican la resistencia del conductor (R), la intensidad de la corriente (I) y el tiempo transcurrido (T).

AMBITO DE UTILIZACION

Una vez definida una función, ésta será accesible durante el resto del programa. Por este motivo, es conveniente que la ejecución sea por las líneas de definición al principio del programa. Desde ese preciso instante ya serán utilizables. E incluso al detenerse la ejecución del programa puede hacerse uso de ellas en modo directo, fuera del programa. Ello supone que su definición sigue siendo válida. Sin embargo, las funciones definidas dentro de un programa no pueden ser utilizadas en otro distinto. La razón se encuentra en el hecho de que la ejecución de un comando RUN borra tanto las variables como las funciones definidas con anterioridad. En consecuencia, para que sea posible hacer uso de las funciones en un segundo programa será necesario volver a definirlas dentro del nuevo programa.

Pascal (10)

Registros y archivos

Analizadas ya las ventajas de la estructuración de los datos y estudiados los dos primeros tipos de datos estructurados —matrices y conjuntos (ARRAY y SET)—, sólo nos quedan por examinar los dos últimos tipos: registros y archivos o ficheros.

El tipo *registro* (RECORD) es el más flexible de todos, ya que permite reunir datos de cualquier otro tipo, incluso nuevos registros, dentro de una misma estructura.

La denominación de "registro" está presente con harta frecuencia en la vida cotidiana, dentro de frases como: registro de empleados, oficina de registros, registro de la propiedad..., lo que da una una idea de una gran utilidad.

En PASCAL, el concepto de registro se refiere a un conjunto de informaciones relativas a un único objeto, que pueden ser utilizadas como un "todo agrupado", o referenciando a sus partes individualmente. Estas partes, denominadas *campos,* mantienen una cierta independencia unas respecto a otras, lo que permite su diferenciación clara en tipo y contenido.

A título de ejemplo, se puede pensar en la información reflejada en un carnet de identidad que constituye, por sí mismo, un "registro" con la identidad de una persona:

— Número de orden
— Nombre
— Apellidos
— Lugar de nacimiento
— Fecha de nacimiento
— Nombres de los padres
— Estado civil
— Profesión
— Domicilio
— Grupo sanguíneo

Este registro, que da lugar al carnet de identidad de una persona, está integrado por 10 campos; cada uno de ellos puede contener informaciones cuyo tipo difiere de la incluida en los restantes campos. Obviamente, el tipo de información que se almacene en el campo del nombre del sujeto no tendrá ninguna semejanza con el contenido del campo de la fecha de nacimiento. Este último estará compuesto, a su vez, por otros tres datos que indicarán el día, el mes y el año, mientras que aquél contendrá simplemente una cadena de caracteres. Esta es una de las principales diferencias del tipo registro (RECORD) respecto al tipo ARRAY. La otra diferencia importante reside en que mientras en una matriz o array cada uno de los elementos se referencian por un subíndice, en los registros se hace referencia a sus elementos por el nombre de los campos que lo integran.

Un poco más formalmente, se puede definir un registro en PASCAL como una colección de un número fijo de componentes denominados "campos", estando cada campo definido por un identificador y su tipo de dato correspondiente (tipo que puede ser distinto para cada campo). La definición de un dato de tipo registro se realiza de la siguiente forma:

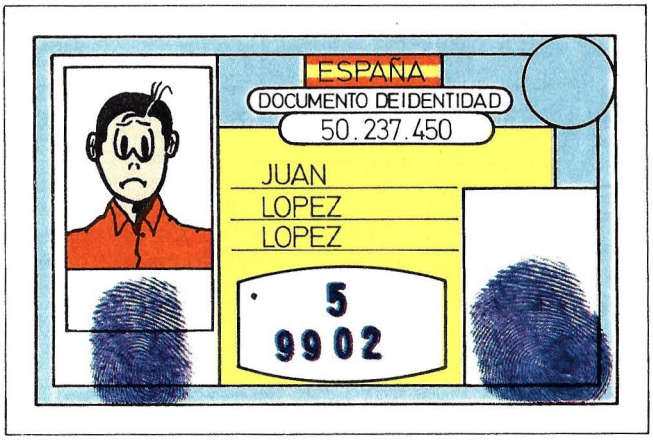
```
TYPE <identificador>=RECORD
       <lista de campos>
       END;
```

Donde la lista de campos estará compuesta por un número finito de identificadores de campo, seguidos por el tipo de cada campo —éste puede ser cualquier tipo estándar o definido con anterioridad—, y separados por el signo de punto y coma. Esta es, ni más ni menos, la estructura genérica:

```
<identificador campo 1>:<tipo>;
<identificador campo 2>:<tipo>;
...
<identificador campo n>:<tipo>;
```

Veamos un ejemplo:

```
TYPE fecha=RECORD
       día: (1..31);
       mes: (ENE, FEB, MAR, ABR, MAY,
       JUN, JUL, AGO, SEP, OCT, NOV,
       DIC);
       año: (1900..2000)
       END;
TYPE complejos=RECORD
       partereal:REAL;
       parteimag:REAL
       END;
```



El tipo RECORD (registro) permite crear una estructura con varios "campos" o componentes que pueden ser de muy diversos tipos.

Las palabras "fecha" y "complejos" son los identificadores de los registros definidos, mientras que "día", "mes", "partereal", etc., son los identificadores de los campos.

SELECCIONANDO LOS CAMPOS

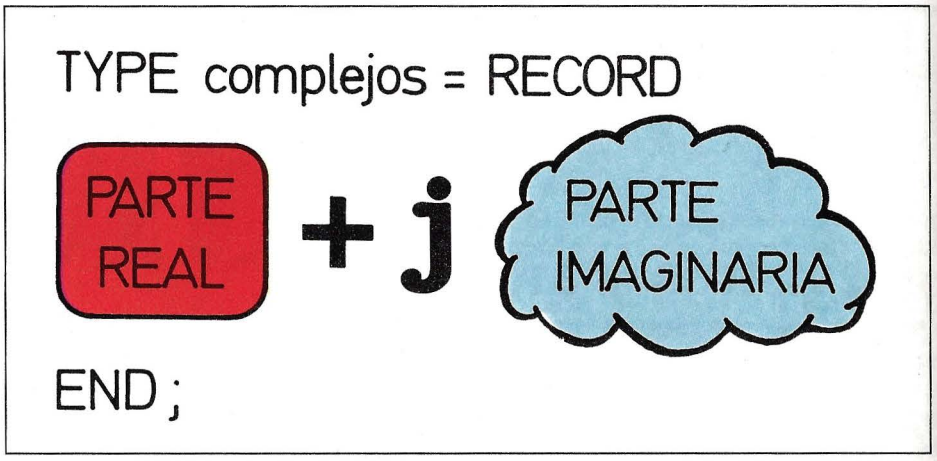
Como ya se ha mencionado, los elementos de un registro se referencian mediante el identificador de los campos, a través del llamado "selector de campo". Este no consiste más que en el identificador de la variable RECORD seguido por un punto y por el identificador del campo.
Un registro puede utilizarse tanto en definiciones de tipo como en declaraciones de variables; así, si se declara la variable "fechanacim" como del tipo "fecha" (definido en el ejemplo anterior) mediante la correspondiente sentencia VAR, se podrá hacer referencia a sus campos respectivos mediante los siguientes selectores de campos:

```
fechanacim.día
fechanacim.mes
fechanacim.anyo
```

De esta forma, cualquier campo de un registro puede ser utilizado como si se tratara (de hecho lo es) de una variable normal. Pero no sólo se pueden utilizar los campos por separado, sino que además se puede hacer referencia al registro en bloque, ya sea dentro de sentencias de asignación, como argumento de funciones, etc. Veamos el siguiente ejemplo:

```
TYPE complejos=RECORD
                 partereal:REAL;
                 parteimag:REAL
       END;
VAR reactancia: complejos;
    inductancia: complejos;
BEGIN
reactancia.partereal:=0;
reactancia.parteimag:=0;
inductancia:=reactancia;
...
END.
```

La última sentencia de asignación copiará en su integridad el registro "reactancia" en el registro "inductancia". El efecto sería el mismo si se tratara de registros con



Las estructuras de tipo RECORD resultan de gran ayuda en múltiples situaciones; por ejemplo, cuando hay que trabajar con números complejos.

mayor número de campos y con los tipos de los campos más complicados (estructurados).
Una sentencia que ayuda a manejar los registros es WITH, cuyo formato es el siguiente:

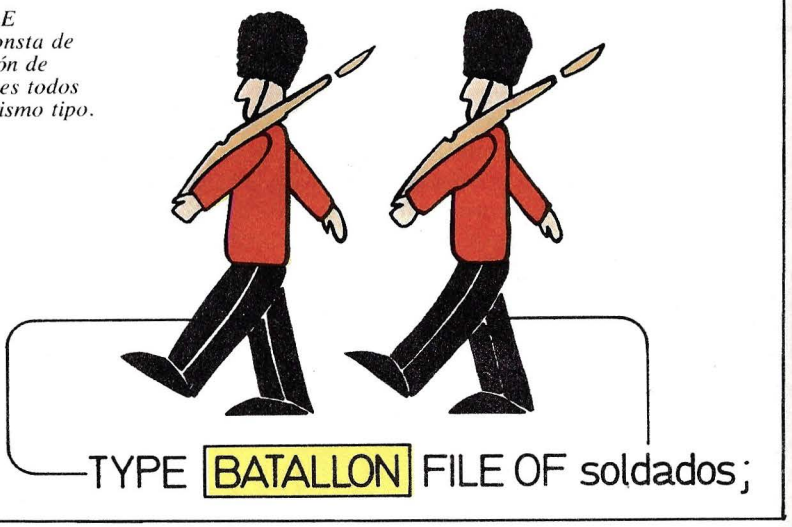
WITH <variable registro> DO <sentencia>;

Con ella se puede omitir el identificador del registro, indicando únicamente el identificador del campo que se desea seleccionar. De esta forma se abreviará sustancialmente la escritura de un programa, como puede observarse en el siguiente ejemplo:

```
WITH reactancia DO
       BEGIN
             partereal:= 4;
             parteimag:= -3.9;
             modulo:=SQRT(SQR(partereal)+SQR(parteimag));
             WRITELN(modulo)
       END;
```

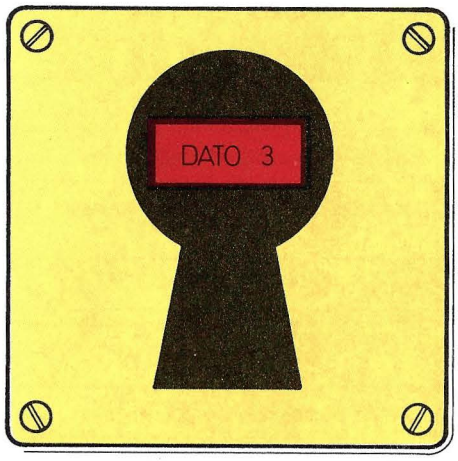
La sentencia WITH sirve de gran ayuda en el caso de tener registros de estructura compleja, en los que alguno de sus campos sean a su vez nuevos registros (registros jerarquizados).

El tipo FILE (fichero) consta de una sucesión de componentes todos ellos del mismo tipo.



LOS ARCHIVOS DE DATOS

En muchas ocasiones resulta útil disponer de medios para crear un conjunto de da-



La variable "buffer" de un fichero se utiliza a modo de "ventana" a través de la cual se puede leer o escribir un elemento del fichero.

tos que pueda ser utilizado en otro momento por el mismo ordenador, o incluso por otro ordenador diferente. Este conjunto de datos se suele almacenar en los llamados "archivos" o "ficheros": estructuras formadas por una secuencia de componentes, todos del mismo tipo, de forma que sólo se puede acceder a un único componente del mismo en un momento dado. El acceso a un elemento se efectúa recorriendo previamente y en secuencia los restantes elementos que estén por delante del afectado. Lo indicado equivale a trabajar con una estructura secuencial en la que importa el lugar físico en el que se almacenan los componentes y que, generalmente, será una memoria de masa externa (cinta magnética o disco).

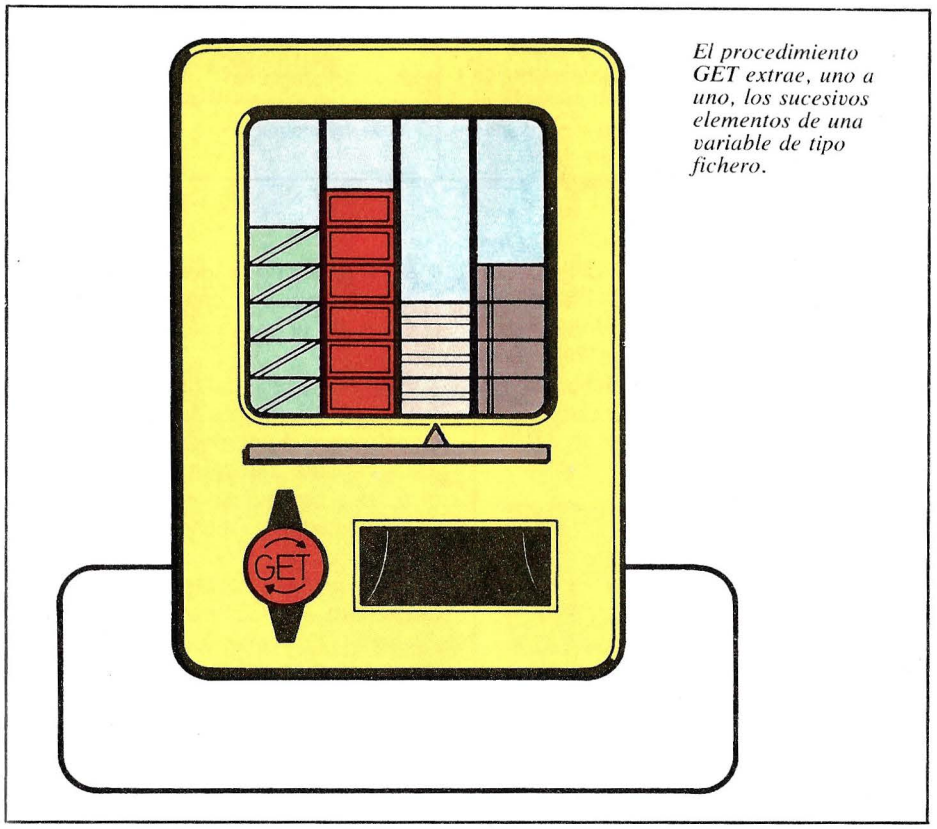
Al hablar de los ficheros estándar de entrada y salida, se comentó la forma general de definir las variables de tipo fichero. Esta era, como se recordará:

TYPE <identificador>=FILE OF <tipo de los elementos>;

Por ejemplo:

VAR numeros=FILE OF complejos;

En donde el tipo de los elementos puede ser cualquiera, incluso un tipo estructurado que haya sido previamente definido. Las variables de tipo fichero sólo se pueden utilizar como parámetros de procedimientos, pero no en sentencias de asignación ni de otro tipo. Para facilitar el manejo de los ficheros existen pues, una serie de procedimientos estándar definidos previamente por el compilador de PASCAL.



El procedimiento GET extrae, uno a uno, los sucesivos elementos de una variable de tipo fichero.

Uno de estos procedimientos es el que tiene por sentencia de llamada: RESET (<identificador de fichero>). Su misión es inicializar el fichero en su primera componente para que pueda ser leída. La lectura podrá realizarse entonces mediante la ya conocida sentencia READ(<identificador de fichero>,x). El valor contenido en la componente actual del fichero pasa a almacenarse en la variable auxiliar "x"; ésta debe ser, por tanto, del mismo tipo que las componentes del fichero. El valor de dicha componente estará ahora disponible en la variable "x" para utilizarlo en la forma que se desee. Para acceder a la siguiente componente se empleará una nueva sentencia READ, y así sucesivamente hasta que se llegue a la última componente del fichero, en cuyo caso, una nueva lectura con READ producirá un error debido a que no existen más componentes. La condición de fin de fichero se puede examinar previamente, para evitar estos errores, mediante otro procedimiento estándar, cuya sentencia de llamada es EOF (<identificador de fichero>). Este procedimiento devuelve el valor booleado TRUE si se ha llegado al final del fichero; de lo contrario, el valor devuelto será FALSE.

La lectura de todas las componentes del fichero "numeros", previamente declarado, se puede programar de la siguiente forma:

```
WHILE NOT EOF(numeros) DO
      BEGIN
            READ(numeros,x);
            ...
      END;
```

Para inicializar la escritura de datos en un fichero se dispone del procedimiento REWRITE (<identificador de fichero>); cuya misión es sustituir el valor del fichero indicado por un nuevo fichero vacío y posicionarse en la primera componente. La escritura se realizará entonces mediante una sentencia WRITE (<identificador de fichero>,x); habiendo asignado previamente a la variable auxiliar "x" el valor que se quiere introducir en la componente actual del fichero. Así, sucesivamente, se irán almacenando todas las componentes del mismo.

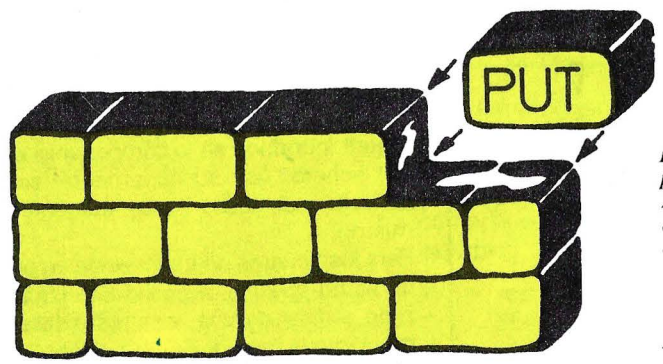
Para los ficheros de texto existe un procedimiento que indica cuando se ha alcanzado el final de una línea; éste se activa mediante la llamada EOLN (<identificador de fichero>) y devuelve un valor de tipo

TABLA DE COMANDOS-PASCAL (1)		
COMANDO	COMETIDO	OBSERVACIONES
TYPE <id.>=RECORD<lista> END;	Definición del tipo registro	Zona de declaraciones y definiciones
<id.1>:<tipo>; ...; <id.n>:<tipo>;	Definición de los campos de un registro (lista de campos).	Zona de declaraciones y definiciones
<id. registro> .<id. campo>;	Selector de campos del registro	Cuerpo del programa
WITH <var.> DO <sent.>;	Manejo de campos sin utilizar el identificador del registro	Cuerpo del programa
TYPE <id.>=FILE OF <tipo>;	Definición del tipo fichero	Zona de declaraciones y definiciones
RESET (<fichero>);	Inicializar fichero para lectura	Cuerpo del programa
EOF (<fichero>);	Indicador del fin de fichero	Cuerpo del programa

NOTA: <id.>: identificador válido PASCAL. <lista>: lista de campos. <tipo>: cualquier tipo de datos, incluso estructurado. <var.>: variable tipo registro. <sent.>: sentencia simple o compuesta. <fichero>: identificador de una variable tipo fichero.

TABLA DE COMANDOS-PASCAL (2)		
COMANDO	COMETIDO	OBSERVACIONES
REWRITE (<fichero>);	Inicializar fichero para escritura	Cuerpo del programa
EOLN (<fichero>);	Indicador de fin de línea	Cuerpo del programa
<fichero> ↑ ;	Variable "buffer" de un fichero	Cuerpo del programa
PUT (<fichero>);	Traslado al fichero de la variable "buffer"	Cuerpo del programa
GET (<fichero>);	Traslado a la variable "buffer" de la componente actual del fichero	Cuerpo del programa

booleano de forma similar al procedimiento EOF. La declaración de cada variable de tipo fichero crea automáticamente una variable auxiliar denominada "buffer", con cabida para un único valor del mismo tipo que las componentes del fichero; ésta variable se utiliza como "intermediaria" en la lectura y escritura de valores.
La variable auxiliar "buffer" está a disposición del programador en cualquier momento, apelando al identificador: <nombre del fichero> ↑ ; en ella se almacenará el valor de la componente actual del fichero.
Así, en el ejemplo del fichero "numeros", creado anteriormente, la variable "buffer" se referenciará por medio de: numeros ↑ .
Esta variable también es utilizada por otros dos procedimientos estándar: GET (<identificador de fichero>) y PUT (<identificador de fichero>) que constituyen otra forma de actualizar las componentes del fichero. Así, GET obtiene el valor de la siguiente componente y lo almacena en la variable buffer del fichero, mientras que PUT traslada el valor del buffer a la componente actual del fichero; lógicamente, ello sólo tendrá sentido cuando EOF sea TRUE, debido a la estructura secuencial del tipo fichero.

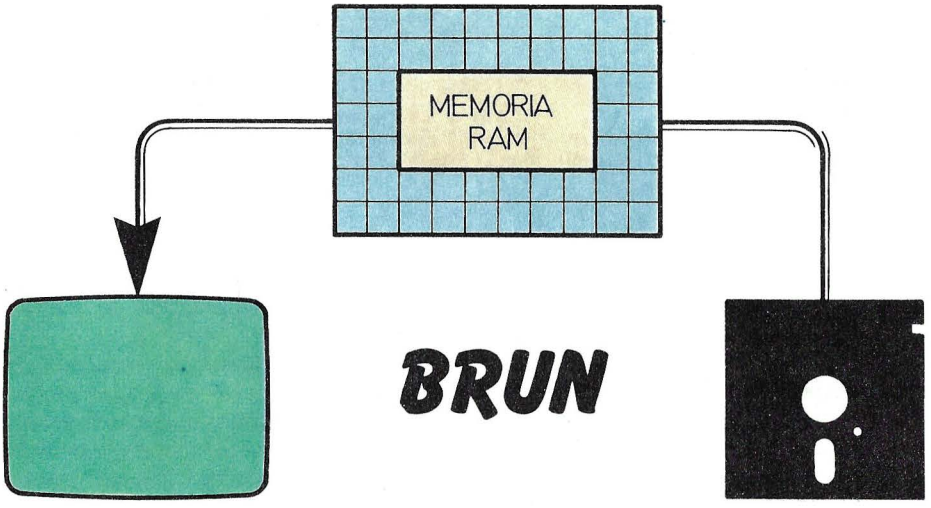


PUT es un procedimiento PASCAL que realiza la función contraria a GET; esto es: introduce un nuevo elemento en la última posición del fichero.

Apple DOS 3.3 (3)

Gestión de la memoria primaria

La familia de microordenadores Apple II está integrada por los clásicos Apple II, Apple II plus y Apple IIe, y con el reciente portátil Apple IIc. Los tres primeros modelos están basados en el microprocesador de ocho bits 6502, cuyo bus de direcciones permite direccionar hasta 65536 posiciones de memoria (64 Kbytes). En la tarea de programación, es costumbre referenciar a estas posiciones de memoria según dos sistemas de numeración: el decimal y el hexadecimal. Para distinguir una notación de otra, y evitar así cualquier posible confusión, suele adoptarse el convenio de preceder a las direcciones expresadas en hexadecimal con el signo dólar "$".



El comando BRUN carga en memoria una imagen de ésta, previamente almacenada en disco; acto seguido, ejecuta el programa en código máquina a partir de la dirección especificada cuando éste fue transferido a disco.

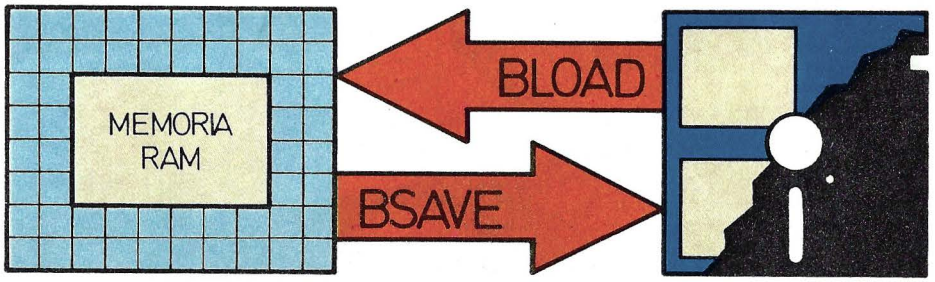
COMANDOS BINARIOS

El método más utilizado para el manejo de la información contenida en la memoria se basa en el traspaso del contenido de las diversas posiciones de la memoria a un archivo en disco, y viceversa. Dicho contenido se manipula byte a byte, expresando a éstos en notación hexadecimal: desde el $00 al $FF (0 a 255 en decimal). Por medio de esta serie de códigos hexadecimales de dos dígitos es posible representar números, letras, comandos en código máquina, información relativa a gráficos, etc.

De este modo, el usuario tiene la posibilidad de almacenar con carácter permanente (no volátil) el contenido de una determinada zona de la memoria interna. Así evitará la pérdida de dicha información en el momento de interrumpir el suministro eléctrico al microordenador. Realizada esta operación, el usuario estará en situación de restaurar en cualquier momento el contenido de la memoria, leyéndolo del fichero en disco en el que fue previamente almacenado.

La primera categoría de ficheros es, pues, la integrada por los ficheros binarios. Estos se distinguen en los listados producidos por el comando CATALOG al ir precedido su nombre por la letra B, indicadora del tipo de fichero. Su utilización está orientada, principalmente, a la ejecución de programas en código máquina y a la salvaguarda de imágenes de la memoria para su uso posterior.

Existen varios comandos del DOS versión 3.3 que actúan sobre estos ficheros, muy parecidos tanto en su sintaxis como en su función a los que se aplican sobre ficheros BASIC.



La transferencia del contenido de la memoria entre ella misma y la unidad de disco, corre a cargo de los comandos BLOAD y BSAVE, similares a los conocidos LOAD y SAVE.

```
] CALL -151 (llamada al monitor)
* 300: 20 80 3F OA.....

* BSAVE PROG, A$300, L$20

* BRUN PROG
```

A

```
] 10 DIR =
] 20 FOR I = 0 TO 31
] 30 READ A
] 40 POKE (768+I), A
] 50 NEXT I
] 60 DATA 20,52,136,251,47,
] RUN

] BSAVE PROG, A$300,L$20

] BLOAD PROG, A$300
] CALL 768
```

B

La figura muestra dos formas de crear, grabar y ejecutar un programa en código máquina: haciendo uso del Monitor (izquierda) y con el BASIC (derecha). Los recuadros del mismo color son equivalentes en su cometido.

● BRUN

Mediante este comando se carga en memoria el contenido de un fichero binario, realizando su lectura de la unidad de disco correspondiente. El sistema operativo coloca los diversos bytes de información leídos de dicho fichero a partir de la dirección especificada cuando fueron grabados en disco. Con ello, se reproduce la situación preexistente, aunque también es posible indicar otra dirección de memoria distinta.

Es aconsejable diferenciar entre los ficheros cuyo contenido son programas binarios y los que están compuestos por datos binarios (por ejemplo con una nomenclatura que los diferencie). Hay que tener en cuenta que el DOS no es capaz de distinguir entre unos y otros; por lo tanto, si se trata de ejecutar un fichero binario cuyo contenido son datos, es muy posible que el sistema operativo quede quebrantado en alguna medida, siendo necesario en tal caso una reinicialización del sistema.

La sintaxis de este comando es la que sigue:

BRUN nf [,An] [,Sn] [,Dn] [,Vn]

nf: nombre del fichero.
An: dirección de memoria a la que se desea transferir el primer byte del programa (de 0 a 65335, ó de $0000 a $FFFF).
Sn: número del conector (1 a 7) que contiene la tarjeta que controla la unidad del disco a la que se accede.
Dn: número de la unidad de disco (1 ó 2) que contiene a nf.
Vn: número de volumen (0 a 255) del disco a ser accedido.
Los argumentos entre corchetes indican que su inclusión no es obligatoria, tomándose los valores existentes por defecto en el caso de no ser especificados.

● BLOAD

Permite la carga desde el disco de un fichero cuyo contenido coincide con una imagen de memoria de cualquier tipo, o con un programa escrito en código máquina. El comando LOAD se distingue por el hecho de que no borra los programas y variables preexistentes, a no ser que caigan en las posiciones de memoria en las que se realiza la carga.
Sintaxis:

BLOAD nf [,An] [,Sn] [,Dn] [,Vn]

● BSAVE

Se comporta de forma inversa al comando anterior: transfiere el contenido de una zona de la memoria a un fichero binario.
Sintaxis:

BSAVE nf,An,Ln, [,Sn] [,Dn] [,Vn]

An: localización del primer byte a transferir (de 0 a 65535, ó de $0 a $FFFF).
Ln: número de bytes a transferir (de 0 a 3276, ó $0 a $7FFF).

La introducción de información binaria en la memoria del ordenador se puede hacer de dos formas. La primera alternativa es desde el programa Monitor, indicando cuál es la dirección a partir de la cual se

Los equipos de la familia Apple II, explotables bajo el control del sistema operativo DOS 3.3, cuentan con grandes facultades para el trabajo con ficheros residentes en disco.

van a emplazar los datos y, a continuación, introduciendo dicha información. La segunda es desde un programa BASIC, recurriendo a la sentencia POKE para que acomode el contenido en las adecuadas posiciones de memoria.
Así pues, para la creación, grabación en disco y posterior ejecución de un programa en código máquina, es necesario utilizar el Monitor o un programa BASIC para crear el programa, el comando BSAVE para grabarlo y, posteriormente, el comando BRUN para cargarlo y ejecutarlo; o bien, el comando BLOAD para cargarlo y la instrucción CALL para ejecutarlo (ver figura).

MAPA DE MEMORIA

Posiciones de HIMEM

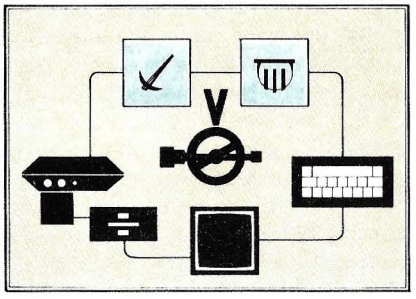
Capacidad de RAM	Posición superior de RAM		Posición HIMEM	
	Decimal	Hex.	Decimal	Hex.
16K	16383	$3FFF	5632	$1600
20K	20487	$4FFF	9728	$2600
24K	24575	$5FFF	13824	$3600
32K	32767	$7FFF	22016	$5600
36K	36863	$8FFF	26112	$6600
48K	49151	$BFFF	27136	$9600

Al cargar el sistema operativo DOS 3.3 éste otorga un valor a la posición HIMEM dependiendo de la capacidad de memoria RAM del sistema.

La memoria de acceso aleatorio (memoria RAM) a disposición del usuario no es en ningún momento la cantidad nominal expresada por el fabricante, sino que la zona útil está comprendida entre los límites definidos por dos direcciones extremas: LOMEM y HIMEM; la primera define el umbral inferior y la segunda el superior.
LOMEM es la dirección de memoria más baja utilizable por el usuario. Esta constituye el origen para el almacenamiento de líneas de programas en BASIC Applesoft y para el almacenamiento de variables en ambos lenguajes BASIC (Applesotf e Integer). El límite definido por LOMEM va desplazándose hacia posiciones más altas de memoria según se van introduciendo las diversas líneas y variables del programa. Esta posición queda desplazada automáticamente a 1024 direcciones más arriba en el caso de que se utilice la página 2 para caracteres y gráficos de baja resolución. Por debajo de LOMEM quedan la página 1, utilizada para la definición de caracteres y de gráficos de baja resolu-

Control de procesos por microordenador

Un área que en los últimos años goza de constante expansión es la relativa al control de procesos industriales por ordenador. Esta es una tendencia cada vez más acelerada debido, en gran medida, a que los sistemas basados en microprocesador presentan tiempos de respuesta cada vez menores y, además, ofrecen al usuario una mayor flexibilidad en sus acciones.
Un ejemplo clásico cabe encontrarlo en la siderurgia; sin ir más lejos, en el conjunto de ordenadores encargados del control de los procesos de forjado directo del acero, a la salida de los hornos. Estos son procesos que requieren un control de la producción en tiempo real con acceso a múltiples datos; todo ello destinado a evitar que sea necesario el almacenaje a alta temperatura del material, con sus costes añadidos, y a obviar la posibilidad de que se produzca un corte en el suministro de material a los equipos por un defectuoso control de la capacidad de producción.
En el campo del control automatizado, la capacidad creciente de los microordenadores y, muy especialmente, el hecho de disponer de sistemas de almacenamiento masivo de información cada vez más perfeccionados y fiables, está dando pie a una fuerte irrupción de los ordenadores. Su utilidad se manifiesta, principalmente, en la manipulación y

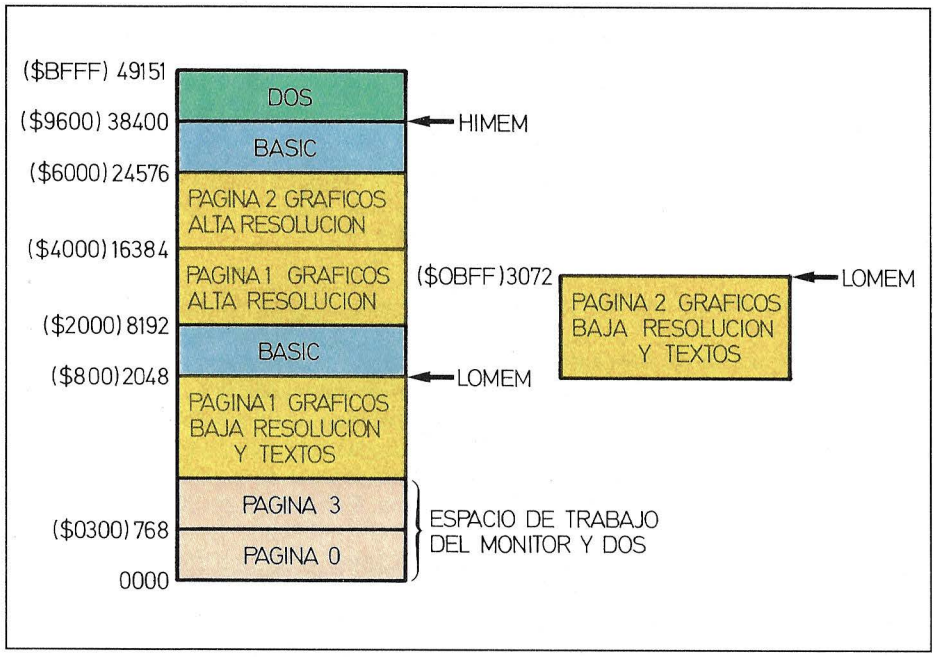


almacenamiento de información relativa a aspectos discretos de un proceso productivo; la gran capacidad de los actuales sistemas para el tratamiento de información en forma gráfica hace que su aplicación sea particularmente atractiva. En la actualidad, y basados en equipos tan simples (relativamente) como puede ser un Apple II, se encuentran en operación sistemas que controlan la variación de más de una docena de variables de proceso distintas. Equipos que suministran al personal técnico de la planta una información pormenorizada e inmediata de dichas variables (valores actuales, máximos, mínimos, medios... a lo largo de un período determinado).
La presencia de la microinformática está llegando a lugares cada vez más distantes de su concepción inicial, haciendo sentir sus beneficios en cualquier ámbito de actividad.

PARAMETROS DE CALL Y G PARA ACTIVAR EL DOS 3.3

Capacidad de memoria	CALL	G (Monitor)
48K y mayores	−25153	$9DBF
32K	23999	$5DBF
16K	7615	$1DBF

NOTA. −25153 corresponde a 40383. Su representación en modo negativo obedece a que el BASIC Integer no admite números superiores a 32767.



Mapa de memoria correspondiente a un equipo Apple II dotado de 48 Kbytes de memoria RAM.

ción, y la página 2 destinada a la extensión de las definiciones anteriores en el caso de que así se especifique. Ambas páginas tienen una extensión de 1024 posiciones de memoria y se utilizan para definir el juego de caracteres disponibles y las características de los puntos de la pantalla en los gráficos de baja resolución.

Por último, en las posiciones más bajas de la memoria RAM (de $0 a $03FF, ó de 0 a 1023) está reservada una zona denominada página 0, utilizada como espacio de trabajo por el Monitor, el BASIC y el sistema operativo.

La definición de HIMEM, o posición de memoria más alta en la que un programa BASIC puede almacenar sus sentencias y variables, depende de la presencia o de la ausencia del sistema operativo en la memoria central. En el caso de que no se haya cargado en memoria el sistema operativo, HIMEM apunta a la dirección de memoria RAM más alta disponible en el microordenador. No obstante, si se ha realizado la carga del sistema operativo, esta posición varía; en efecto, ha de quedar por debajo de la zona ocupada por el sistema operativo, en orden a evitar que un programa BASIC pueda escribir en esta zona y destruir su contenido.

El posicionamiento de este límite depende de la cantidad de memoria RAM disponible en el ordenador. Su situación queda especificada en las posiciones de memoria 76 y 77 ($4C y $4D) si el BASIC activo es el Integer, o en las posiciones 115 y 116 ($73 y $74) en el caso de que lo esté el BASIC Applesoft. Un dato importante es que siempre hay que dejar un espacio libre entre la posición más alta de la RAM y la posición HIMEM, para que sea posible albergar al sistema operativo. Esta zona libre debe ser, al menos, de 10,5 Kbytes.

El emplazamiento de HIMEM desciende asimismo si se incrementa el número máximo de ficheros que pueden estar abiertos simultáneamente (con el comando MAXFILES). En tal caso, es necesario un espacio de almacenamiento temporal o "buffer" de 595 bytes por cada fichero, lo que incrementa el valor existente por defecto.

De esta forma, el área comprendida entre LOMEM y HIMEM quedaría en principio disponible para los programas BASIC del usuario, aunque esto no es del todo cierto. En efecto, si se utilizan gráficos de alta resolución es necesario reservar un espacio adicional de 8 Kbytes para cada página o zona de memoria en la que se definen los atributos de los puntos de la pantalla en alta resolución; este número de páginas puede ser uno o dos, según determine el usuario.

El uso de gráficos de alta resolución en un equipo con menos de 32 Kbytes de memoria, es incompatible con la presencia simultánea del sistema operativo, ya que ambos se ven obligados a compartir una misma zona de memoria.

PUNTOS DE ENTRADA AL DOS

La porción de memoria comprendida entre las posiciones $0300 y $03FF (768 y 1023 en decimal), también denominada página 3, contiene punteros y subrutinas que utiliza el DOS para su funcionamiento interno. Ello significa que si por algún motivo se activase accidentalmente el programa Monitor, esta zona de memoria quedaría inaccesible, quebrantándose el funcionamiento del sistema operativo. Para volver a activar el DOS no es necesario realizar un arranque en caliente del sistema, sino que basta con ejecutar la sentencia CALL o el comando G perteneciente al monitor; el parámetro a utilizar depende, en cada caso, del tamaño de la memoria del sistema (ver tabla adjunta).

Multiplan (y 3)

Sesión de trabajo con la hoja electrónica

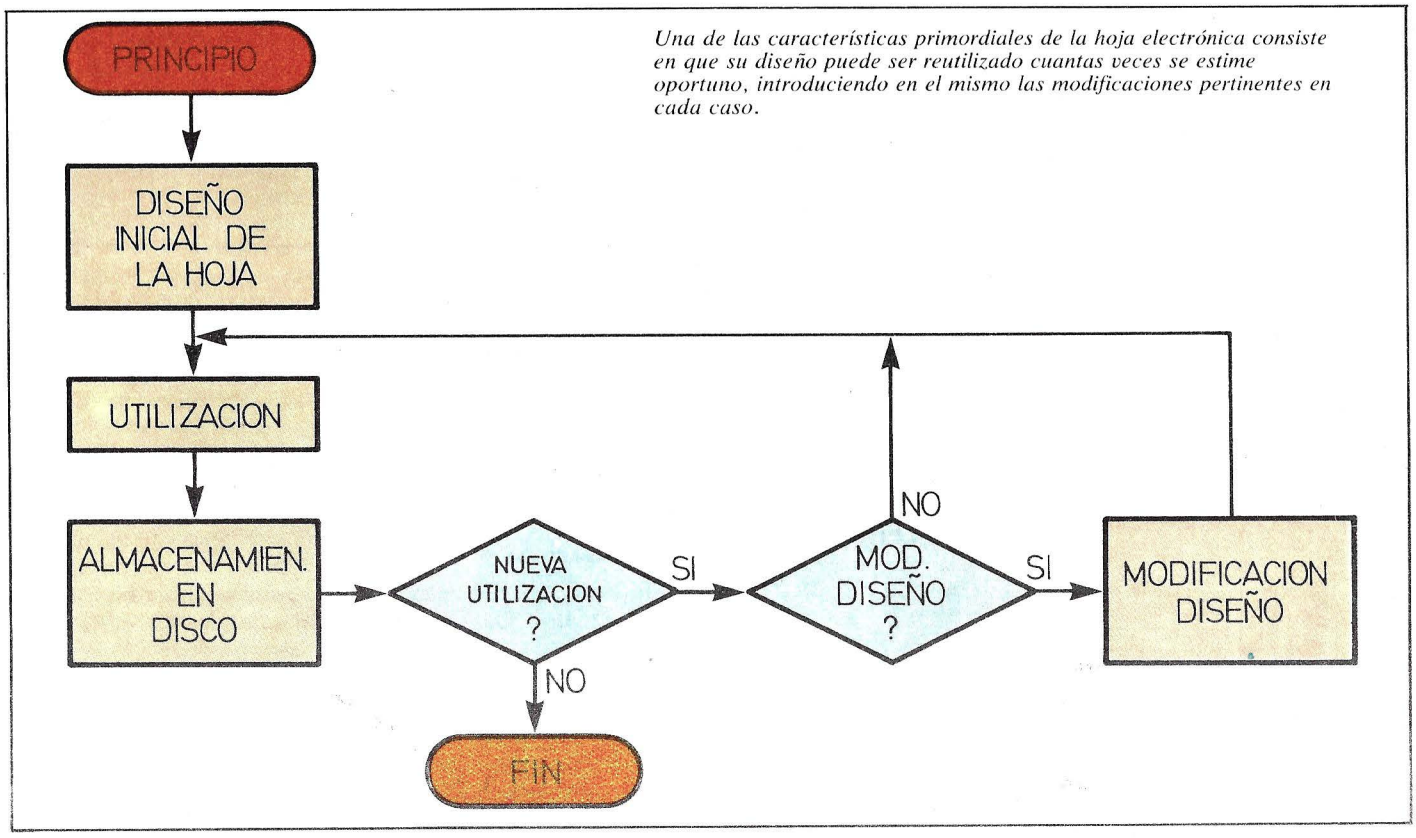
Con ocasión del estudio de la hoja electrónica VISICALC, se resolvieron un par de supuestos prácticos en los que se apreciaba la gran utilidad de este tipo de programas. En este caso, para examinar el funcionamiento práctico del MULTIPLAN, vamos a acometer un ejercicio algo más complejo que los anteriores. Se trata de diseñar una hoja electrónica para confeccionar la declaración del impuesto sobre la renta de las personas físicas (IRPF).

Por la naturaleza de dicho impuesto, el diseño realizado habrá de ser modificado cada año, con toda seguridad; al menos mientras no se consiga alcanzar una ley impositiva estable y duradera. Como especificaciones iniciales para el diseño, hemos elegido en este caso las elaboradas por el Ministerio de Hacienda para la declaración sobre la renta de 1983 (realizada en 1984).

JUSTIFICACION

Antes de entrar de lleno en el caso que nos ocupa, tal vez sea interesante demostrar que una hoja electrónica puede ser una herramienta útil para facilitar la cumplimentación del IRPF. Desde luego, de inmediato podemos encontrar dos argumentos en contra:

1. Como ya se apuntó, el diseño debe ser distinto cada año, puesto que lo más probable es que cada año varíe la ley. Esta circunstancia supone una escasa perdurabilidad de la labor de "programación de la hoja electrónica".
2. El empleo de la hoja de cálculo será puntual, ya que la declaración tan sólo se realiza una vez al año; en consecuencia, puede resultar baldío el esfuerzo de preparar la hoja, al ser éste casi superior al necesario para confeccionar la declaración de forma manual.



Una de las características primordiales de la hoja electrónica consiste en que su diseño puede ser reutilizado cuantas veces se estime oportuno, introduciendo en el mismo las modificaciones pertinentes en cada caso.

Si reflexionamos con calma sobre los referidos argumentos del uso de la hoja electrónica, llegaremos a la conclusión de que, en realidad, más que inconvenientes resultan dos motivos básicos para recomendar su empleo:

1. Efectivamente, el diseño variará cada año y resultará imprescindible modificarlo para que se ajuste a las nuevas normas; precisamente por ello resulta necesario utilizar una hoja electrónica.
En una hoja electrónica, las modificaciones puede realizarlas cómodamente el propio usuario. Sin embargo, de utilizar un programa específico para dicha finalidad, las modificaciones serían más penosas y, probablemente, habría que contar con la participación de un experto en informática.

2. También es cierto que la declaración del IRPF se realiza una única vez al año, y que es posible realizarla "a mano" en el mismo tiempo que se invierte en el diseño inicial de la hoja. Pero en este razonamiento falta un punto muy importante: la declaración sólo se cumplimenta una vez, pero previamente es necesario realizar innumerables tanteos para minimizar el resultado o importe a abonar.
Si los cálculos se efectúan manualmente, el número de pruebas y combinaciones realizadas bajará considerablemente y, en consecuencia, aumentará el riesgo de no encontrar la "combinación ideal" que nos permita pagar estrictamente lo "justo y necesario".
En consecuencia, cabe afirmar que la declaración del IRPF es un problema idóneo para acometer su resolución mediante una hoja electrónica... por ejemplo, con el programa MULTIPLAN.

LAS REGLAS DEL JUEGO

En principio podemos convenir que existen dos partes relativamente autónomas en el impreso de liquidación;

1. Cálculo de la cuota íntegra.

Mediante la suma de los rendimientos netos obtenidos y los incrementos del patrimonio, menos las deducciones por disminuciones, podemos obtener la base imponible. Y, a partir de ésta, aplicando el porcentaje apropiado, obtendremos la cuota íntegra a abonar a la Hacienda Pública.

2. Cálculo de la cuota líquida.

Afortunadamente, la cuota íntegra puede verse disminuida al tener en cuenta las posibles deducciones aplicables. Por lo tanto, el siguiente paso consistirá en calcular la suma de todas las citadas deducciones para que, restando esta cantidad a la cuota íntegra, podamos obtener la cuota líquida. Esta coincidirá con la cantidad que debemos aportar al Erario Público. El último paso será ya el de comprobar si las retenciones realizadas a cuenta son inferiores o superiores a la cuota líquida: si son inferiores, habrá que ingresar la diferencia; mientras que si el importe de las retenciones es superior, habrá que solicitar la devolución de la diferencia.

En ciertas aplicaciones, el uso de una hoja electrónica puede resultar ventajoso respecto al empleo de un programa diseñado al efecto; las razones hay que buscarlas en la versatilidad y fácil redefinición del diseño de la hoja electrónica.
Esta propiedad resulta fundamental en el ejemplo que nos ocupa: preparar la declaración del IRPF.

DISEÑO DE LA HOJA ELECTRONICA

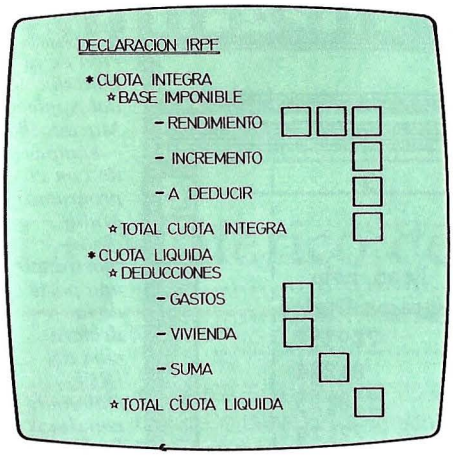
Con objeto de finalizar el ejemplo en este mismo número, vamos a resumir su exposición a los puntos más significativos; no obstante, en las figuras se puede apreciar el aspecto completo del diseño realizado.

Literales

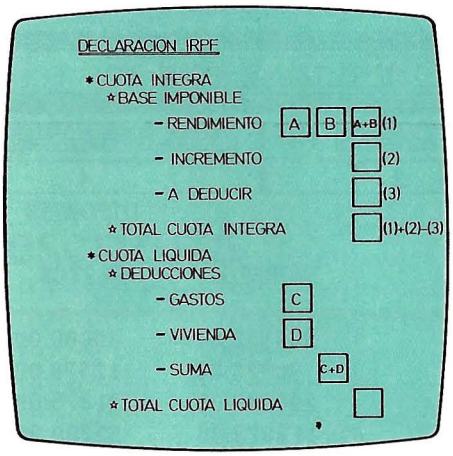
Si suponemos que el ordenador se encuentra ya conectado, y con el programa MULTIPLAN cargado en la memoria principal, el primer paso será, sencillamente, incluir todos los datos fijos en la hoja electrónica. En nuestro caso, y atendiendo a las dos partes fundamentales de la declaración, incluiremos en la primera línea el literal siguiente:

- "Cuota íntegra"

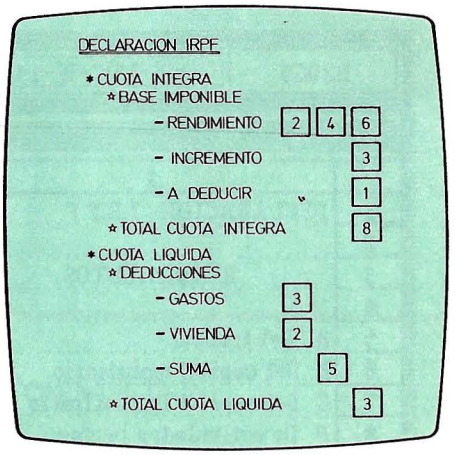
La parte superior del documento original destinado a obtener la cuota íntegra, hace referencia a datos personales del declarante y de su cónyuge. Como estos datos resultan fijos y no influirán en los cálculos, se omitirán en nuestro diseño.
El primer proceso a realizar, dentro de esta zona de la declaración, consiste en el cálculo de la base imponible; ello puede especificarse como literal fijo, desplazándolo a la derecha para reflejar su pertenencia a la cuota íntegra. Dentro de este capítulo cabe distinguir tres subcapítulos: rendimientos, incrementos de patrimonio y cantidades a deducir. De nuevo, desplazando el literal de base imponible a la derecha, incluiremos en la hoja electrónica estos tres literales.
Cada subcapítulo debe dividirse, a su vez, en distintos apartados. En el caso de los rendimientos incluiremos un literal para rendimientos de trabajo, otro para los del capital mobiliario, etc. Los incrementos del patrimonio se subdividirán en tres



La primera fase del diseño de la hoja electrónica consiste en preparar los literales fijos.



La segunda fase del diseño consiste en definir las fórmulas que relacionan a los distintos datos implicados.



La tercera y última fase consiste en la introducción de datos para confirmar que el diseño de la hoja es el correcto.

apartados: Tarifa general, tipo medio y transmisiones lucrativas. Por último, se descompondrán también las posibles deducciones.

Una vez detallados los tres subcapítulos sólo quedarán por incluir cuatro literales: para la identificación de la suma de deducciones; para la diferencia de rendimientos, más incremento del patrimonio, menos deducciones; para las posibles deducciones negativas de años anteriores y, por fin, un literal para la base imponible.

Una vez determinada la base imponible, habrá que calcular la base convertida a la tarifa general; para ello, reservaremos una

Interés compuesto

Las hojas electrónicas pueden aplicarse con eficacia a la resolución de problemas financieros. En este ámbito de aplicación suelen intervenir con frecuencia cálculos de intereses compuestos con distintos períodos de acumulación de los intereses. Veamos cuáles son las fórmulas a utilizar.

Se denomina interés a la ganancia obtenida por el préstamo de un cierto capital, y se dice que el interés es compuesto cuando los intereses producidos en cada unidad de tiempo se agregan al capital primitivo para que produzcan, a su vez, nuevos intereses. Para su cálculo existe una fórmula elemental que permite determinar el capital final (CF) en función del capital inicial (CI), el porcentaje de rendimiento —interés anual producido por una peseta en un año— (PR) y el número de años (NA) que dura el préstamo:

$$CF = CI \cdot (1 + PR)^{NA}$$

En el caso de que la acumulación del interés no sea anual, existen diversas fórmulas alternativas entre las que cabe destacar las siguientes:

- Acumulación semestral:

$$CF = CI \cdot \left(1 + \frac{PR}{2}\right)^{2.NA}$$

- Acumulación trimestral:

$$CF = CI \cdot \left(1 + \frac{PR}{4}\right)^{4.NA}$$

- Acumulación mensual:

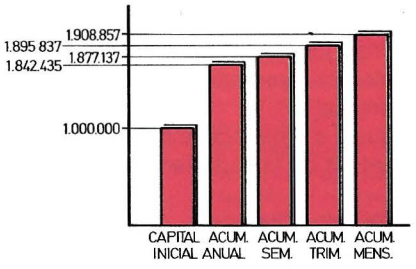
$$CF = CI \cdot \left(1 + \frac{PR}{12}\right)^{12.NA}$$

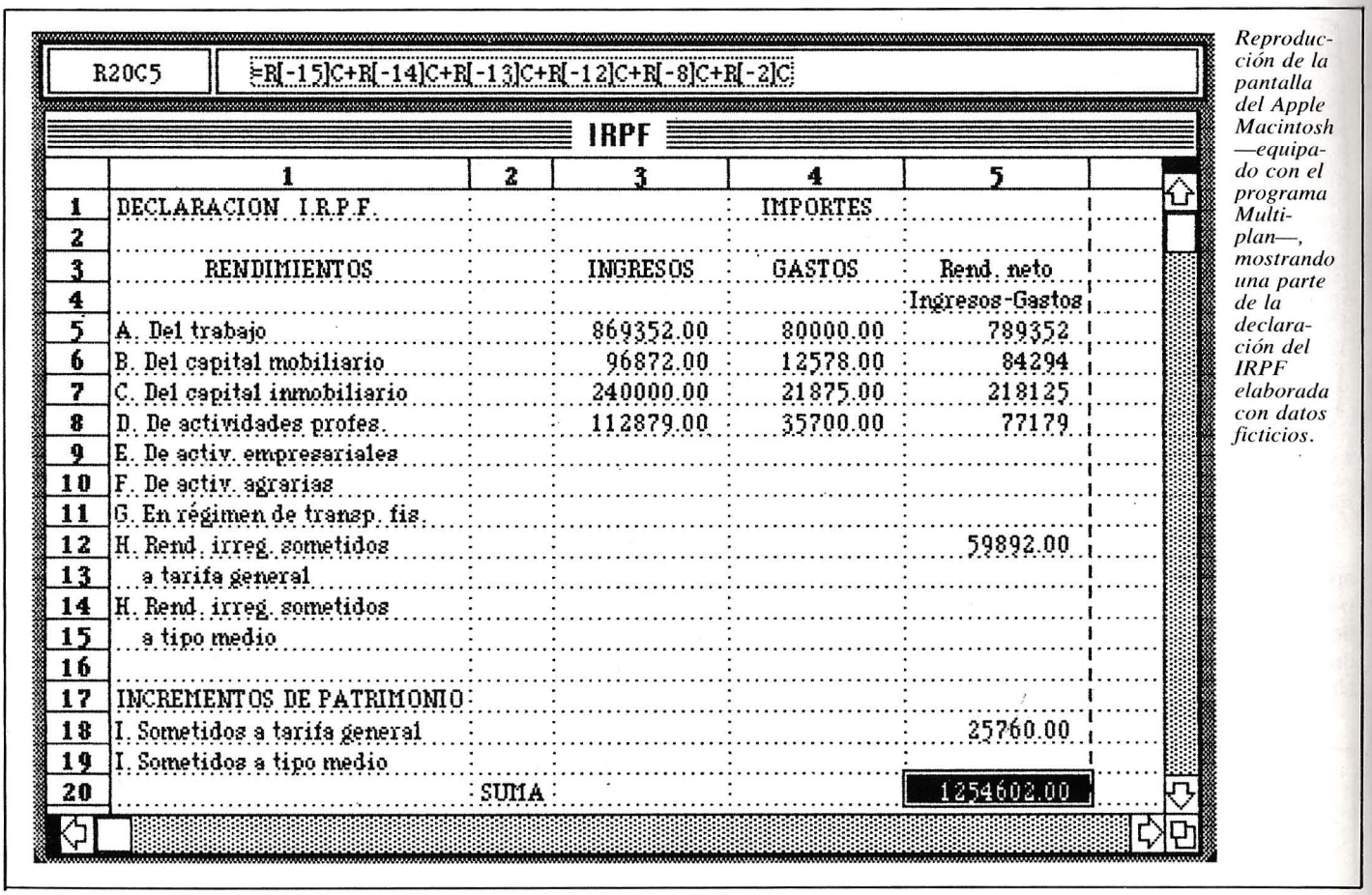
Evidentemente, a medida que disminuye el período de tiempo en el que se realiza la acumulación de los intereses aumenta el capital final. La justificación de esta propiedad estriba en que a medida que se acumulan los intereses éstos a su vez generan nuevos intereses y en consecuencia el capital final a devolver aumenta significativamente.

CAPITAL INICIAL 1.000.000
PORCENTAJE RENDIMIENTO 0,13
NUMERO DE AÑOS 5

CAPITAL FINAL			
ACUMULACION ANUAL	ACUMULACION SEMESTRAL	ACUMULACION TRIMESTRAL	ACUMULACION MENSUAL
1.842.435	1.877.137	1.895.837	1.908.857

Un ejemplo práctico...





R20C5 | =R[-15]C+R[-14]C+R[-13]C+R[-12]C+R[-8]C+R[-2]C

IRPF

	1	2	3	4	5
1	DECLARACION I.R.P.F.			IMPORTES	
2					
3	RENDIMIENTOS		INGRESOS	GASTOS	Rend. neto
4					Ingresos-Gastos
5	A. Del trabajo		869352.00	80000.00	789352
6	B. Del capital mobiliario		96872.00	12578.00	84294
7	C. Del capital inmobiliario		240000.00	21875.00	218125
8	D. De actividades profes.		112879.00	35700.00	77179
9	E. De activ. empresariales				
10	F. De activ. agrarias				
11	G. En régimen de transp. fis.				
12	H. Rend. irreg. sometidos				59892.00
13	a tarifa general				
14	H. Rend. irreg. sometidos				
15	a tipo medio				
16					
17	INCREMENTOS DE PATRIMONIO				
18	I. Sometidos a tarifa general				25760.00
19	I. Sometidos a tipo medio				
20		SUMA			1254602.00

Reproducción de la pantalla del Apple Macintosh —equipado con el programa Multiplan—, mostrando una parte de la declaración del IRPF elaborada con datos ficticios.

única línea con su correspondiente literal.
Para finalizar el cálculo de la cuota íntegra sólo falta indicar la base correspondiente, la cuota correspondiente al escalón anterior y la cuota relativa al resto o diferencia por exceso. Una vez terminados los literales de la cuota íntegra, se empezarán a introducir los correpondientes a la cuota líquida. Para ello, se incluirá el siguiente literal ajustado al margen izquierdo:

- "Cuota líquida"

La forma de ajustar los literales de la hoja electrónica en este segundo apartado es similar a la descrita en el caso anterior. En la correspondiente figura se puede observar el diseño general. En resumen, cabe apuntar que para incluir literales hay que coordinar dos objetivos:

1. Intentar reproducir fielmente el modelo original.
2. Ajuste a la estructura matricial de filas y columnas de la hoja electrónica.

Fórmulas de cálculo

Después de haber diseñado el esqueleto de la hoja electrónica para la declaración de la renta, el siguiente paso debe consistir en la inclusión de las fórmulas con las que se realizarán los cálculos. Para ello, la mejor metodología consiste en ir repasando uno por uno todos los literales y, en cada uno de ellos, decidir si sus valores concretos se introducirán manualmente o si, por el contrario, se pueden calcular en función de otros datos.
Así, por ejemplo, en la primera fila de los rendimientos (del trabajo) se deben introducir manualmente los datos correspondientes a ingresos y gastos; en cambio, el rendimiento neto, es decir "Ingresos" menos "Gastos", puede ser calculado automáticamente, sin más que ordenar que se efectúe una resta. Otro dato que puede ser calculado mediante un sencillo sumatorio es la suma total de los rendimientos más los incrementos del patrimonio.
En el fondo, los cálculos necesarios para la confección de la declaración sobre el IRPF son muy sencillos, tan sólo hay que utilizar las operaciones aritméticas suma y resta y el cálculo de porcentajes.

INTRODUCCION DE DATOS

Una vez finalizado el diseño de la hoja electrónica, ésta se encuentra ya disponible para la entrada de datos. En consecuencia, se debe realizar una introducción inicial de datos con el fin de verificar su buen funcionamiento. Esta comprobación de la eficacia y calidad de la hoja diseñada resulta fundamental para cualquier aplicación; y mucho más en nuestro ejemplo, ya que la existencia de errores en el diseño puede conducir a lamentables consecuencias.
Después de introducir los datos para la prueba, el sistema queda disponible para utilizarlo en todos los tanteos que el usuario estime oportuno realizar. Cuando finalice la sesión de trabajo, hay que obtener una copia en disco de la hoja electrónica para posteriores reutilizaciones.

Impresoras

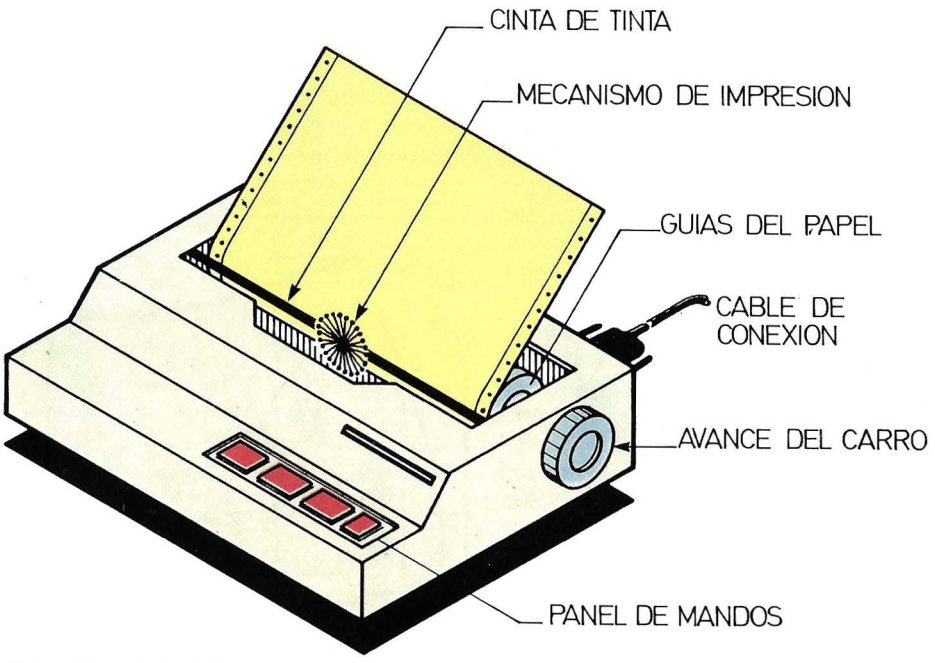
El periférico de escritura

Es posible que muchos usuarios de ordenadores personales no se hayan ni tan siquiera planteado la posibilidad de utilizar una impresora. Realmente, si el ordenador sólo se utiliza con programas de juegos o aplicaciones domésticas sencillas, no necesitará la colaboración de una de estas "máquinas de escribir inteligentes". No obstante, a medida que se adquiere una mayor experiencia en el uso del ordenador, se deseará obtener un mayor partido de sus ingentes posibilidades; ello conducirá a tener en cuenta el empleo de este útil periférico de salida.

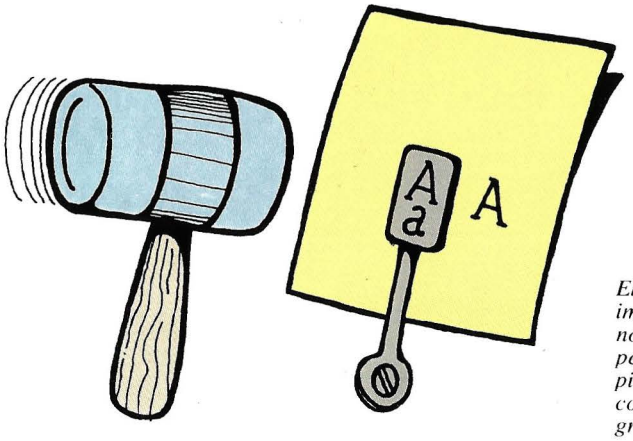
A la hora de confeccionar programas, la colaboración de una impresora resulta de indudable interés; con ella será posible guardar una copia escrita en papel de los listados. El hecho de disponer de copias parciales o totales del programa durante las fases de codificación y depuración resulta de gran ayuda. Su utilidad es extensiva a la obtención de copias impresas de los cálculos realizados por un determinado programa o, sencillamente, de los datos elaborados por un programa, por ejemplo, especializado en la gestión del gasto familiar.

Debido a la gran variedad de modelos que existen en el mercado, la elección por parte del usuario de la impresora adecuada a sus gustos y necesidades debe realizarse con sumo cuidado y conocimiento de causa. Para apoyar tal elección, hay que conocer las características más significativas que es preciso evaluar. Pero vayamos por partes, y empecemos por definir someramente qué es una impresora. Se trata, ni más ni menos, de un periférico de salida que transforma las señales eléctricas procedentes del ordenador en una forma legible para el ser humano; más concretamente, en un documento impreso en papel.

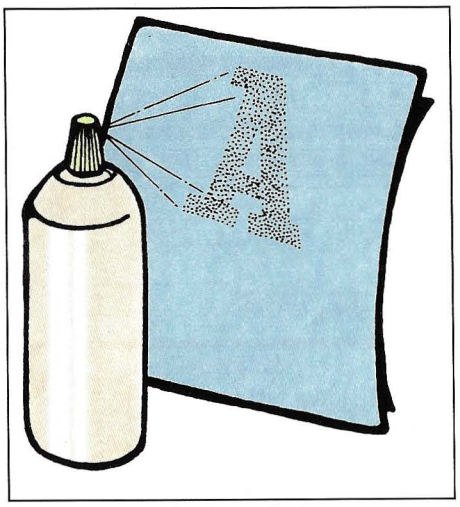
Como se desprende de esta definición, es



Salvo el mecanismo de impresión, los restantes elementos suelen ser comunes a la mayor parte de las impresoras.



El mecanismo de impresión se basa, normalmente, en la percusión de una pieza metálica contra otra que tiene grabado en relieve el carácter a imprimir.



Las impresoras de chorro de tinta proyectan sobre el papel gotas de tinta en la proporción adecuada para "pintar" literalmente el carácter.

el ordenador quien gobierna los movimientos de la impresora. La intervención del usuario tan sólo es necesaria para la correcta conexión de ambas máquinas, y para "dictar" al ordenador las instrucciones necesarias para que éste transfiera la información adecuada a la impresora.

TIPOS DE IMPRESORAS

La principal diferencia entre los distintos tipos de impresoras reside en la técnica empleada en sus cabezales de impresión; éste es, precisamente, el elemento que se encargará de plasmar los caracteres en el papel. Sus restantes elementos constitutivos son análogos; tan sólo muestran pequeñas diferencias en algunos aspectos tales como la forma de arrastre del papel, mandos de control, circuitos electrónicos de gobierno, conexiones con el exterior...

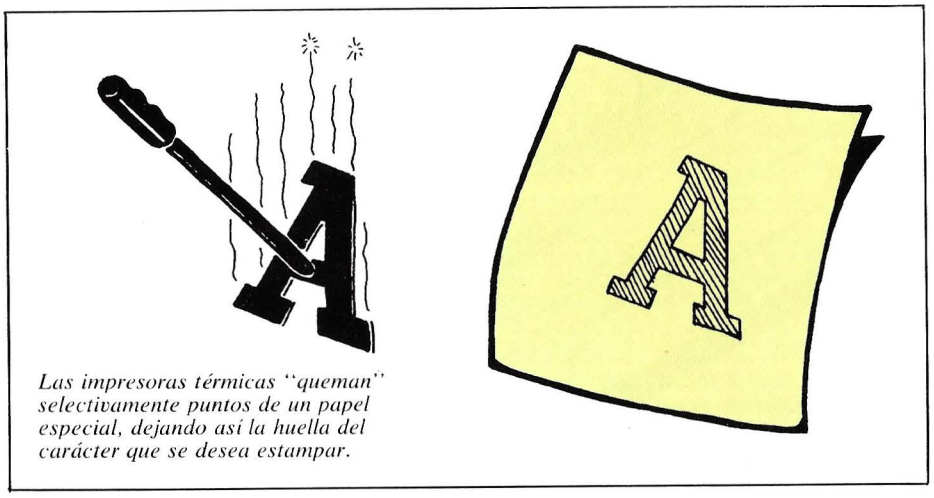
Son tres los tipos de impresoras más frecuentes en el ámbito del microordenador: las matriciales, las de margarita y las térmicas. La impresora matricial basa su funcionamiento en un juego de "agujas" metálicas, situadas en la cabeza de impresión, que forman una cuadrícula (matriz) con la cual se van componiendo los diferentes caracteres. Al mismo tiempo que la cabeza de impresión se va desplazando por el papel, las señales eléctricas que envía el ordenador se encargan de activar esas agujas según corresponda al carácter a imprimir. Las agujas golpean contra una cinta entintada, y ésta sobre el papel, dibujando cada aguja activada un punto en la superficie del mismo.

Mediante la adecuada combinación de estos puntos, se van creando los diversos caracteres que componen un texto. Con este mismo método, es posible imprimir también gráficos y dibujos; ello supone tan sólo codificar correctamente la información que debe activar a las distintas agujas de la matriz.

Las impresoras matriciales constituyen el tipo más corriente en el mundo de los ordenadores personales. Entre sus ventajas cabe citar una buena velocidad de impresión y una moderada economía; sin lugar a dudas, su sistema de impresión resulta sencillo en comparación con el de los restantes tipos de impresoras. Aunque también cabe hablar de inconvenientes; por ejemplo: son bastantes ruidosas y su calidad de impresión suele ser reducida, debido a que las letras y números están formados por una conjunción de puntos estampados en el papel.

Cuando es necesario obtener copias de gran calidad y con una buena presentación, suele recurrirse a las impresoras de rueda de margarita; así llamadas por utilizar como mecanismo de impresión una rueda con pétalos, cuya forma recuerda a la mencionada flor, en cuyos extremos se encuentran situados los diferentes caracteres imprimibles.

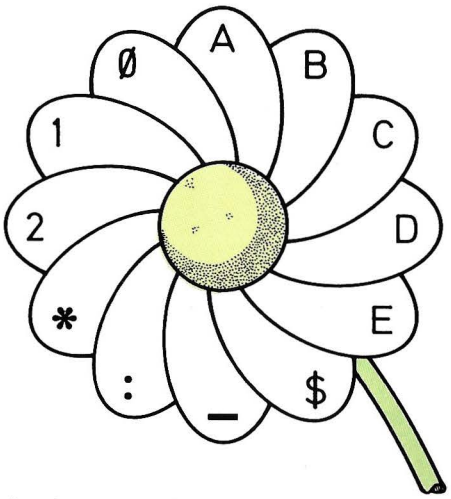
El "modus operandi" de este tipo de impresoras consiste en el giro de la rueda de la margarita y la actuación sincronizada de un "martillo" metálico; éste golpea en



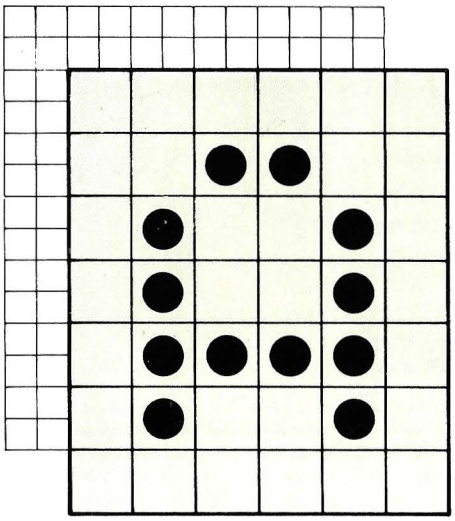
Las impresoras térmicas "queman" selectivamente puntos de un papel especial, dejando así la huella del carácter que se desea estampar.

Las impresoras de matriz de puntos suelen constituir una alternativa muy frecuente para el usuario de un ordenador de tipo doméstico o personal. Su precio moderado y su aceptable calidad abogan por tal elección.

cada caso al "pétalo" que transporta el carácter cuyo código corresponde con las señales eléctricas recibidas del ordenador. El martillo golpea entonces contra el pétalo, la cinta y el papel, dejando huella impresa del carácter en cuestión.
La calidad de impresión que se obtiene con este tipo de impresoras es comparable al de las mejores máquinas de escribir. Esta es su principal virtud, que la hace muy adecuada para aplicaciones tales como el tratamiento de textos, donde se requiere una gran calidad de letra. En muy pocos segundos es posible cambiar la margarita por otra con distinto alfabeto de caracteres, logrando así distintos tipos de impresión.
Como principales inconvenientes cabe señalar que las impresoras de margarita son bastante más lentas que las matriciales; por lo demás, con ellas no es posible crear gráficos o dibujos, pues haría falta una gran cantidad de ruedas diferentes para reproducir los innumerables matices de un dibujo. Otra desventaja es su precio, bastante más elevado que las matriciales.
El tercer tipo de impresoras, extendido en el campo de los microordenadores, lo constituyen las impresoras térmicas. En ellas, los caracteres se plasman por efecto del calor. Este es traspasado desde una cabeza de impresión especial, formada por resistencias eléctricas que se calientan al recibir una señal, a un papel sensible al calor, que se oscurece bajo su influencia. Este método de impresión cuenta con la ventaja de ser muy silencioso, además de bastante rápido y razonablemente económico. A estas ventajas se opone la necesidad de un papel especial, más caro que el normal, y la baja calidad de la impresión.
Existen otros muchos tipos de impresoras; si bien, los mencionados en las líneas anteriores son los más populares y accesibles para el usuario medio de un ordenador personal. Entre los restantes modelos, cabe citar aquí a las impresoras de chorro de tinta, que van "pintando" literalmente los caracteres por medio de gotas de tinta, lanzadas desde la cabeza de impresión y dirigidas mediante campos electromagnéticos. También hay que hacer mención a las impresoras de líneas, que escriben "línea a línea" en lugar de carácter a carácter, con lo que consiguen una mayor rapidez de impresión; y a las de laser, que ponen en práctica sofisticadas técnicas que conducen a un resultado muy veloz y de alta calidad.



Las impresoras de margarita deben su nombre a la similitud que presenta el elemento de impresión con dicha flor. El dispositivo consiste en una rueda en cuyos "pétalos" están situados los caracteres que es capaz de representar.



La formación de los caracteres en las impresoras matriciales tiene lugar por medio de una serie de puntos. Estos son estampados sobre el papel con unos minúsculos martillos —las agujas— que presionan a la cinta entintada sobre el papel al ser activados por las señales procedentes del ordenador.

LA IMPRESORA DESDE EL BASIC

Las impresoras, al igual que los ordenadores, son máquinas incapaces por sí mismas de realizar ninguna acción. De la misma manera que al ordenador hay que adiestrarle por medio de un programa, a la impresora también hay que indicarle las tareas que se desea que realice; pero en este caso es el mismo programa y, por lo tanto, el propio ordenador el que sirve de intermediario entre el usuario y la impresora. Las órdenes que determinarán el funcionamiento de la impresora adoptarán la forma de comandos BASIC.
¿Cómo se puede obtener una copia impresa del listado del programa? Como ya se sabe, el listado del programa se puede visualizar en pantalla o monitor mediante el comando LIST. En buena lógica, la orden adecuada para extraer el listado por impresora no debería diferir mucho de ésta; y así es en realidad. Para tal fin se emplea este mismo comando, aunque con ciertas variantes.
En efecto, es preciso indicar al ordenador, de una u otra forma, a través de qué dispositivo periférico de salida (pantalla, impresora...) se desea la emisión del listado del programa. Como quiera que, generalmente, esto se realiza por pantalla, si tan sólo se formula la orden LIST, el ordenador asumirá que el listado debe visualizarse por pantalla. Para hacerlo a través de la impresora, será preciso indicar este hecho mediante el código correspondiente a este dispositivo que, generalmente, es la letra P (del inglés PRINTER: impresora) encerrada entre comillas. Así, el comando que en muchos dialectos BASIC permite obtener un listado por impresora presenta el siguiente formato:

<nl.> LIST "P",[<nl1.>-<nl2.>]

En el que la expresión <nl1.> indica el número de la primera línea del programa a partir de la cual se desea obtener en listado, y <nl2.> precisa la última línea a listar.
Por ejemplo, si en la memoria del ordenador se encuentra el siguiente programa:

```
10 REM EJEMPLO DE LISTADO
20 CLS
30 PRINT "PULSA UNA TECLA"
40 IF INKEY$="" THEN GOTO 40
50 PRINT "ESA TECLA NO VALE"
60 GOTO 30
```

Al ejecutar la orden LIST "P" se obtendrá un listado del programa completo por impresora; mientras que si la orden introducida es LIST "P", 20-50, el resultado en la impresora será el siguiente:

```
20 CLS
30 PRINT "PULSA UNA TECLA"
```

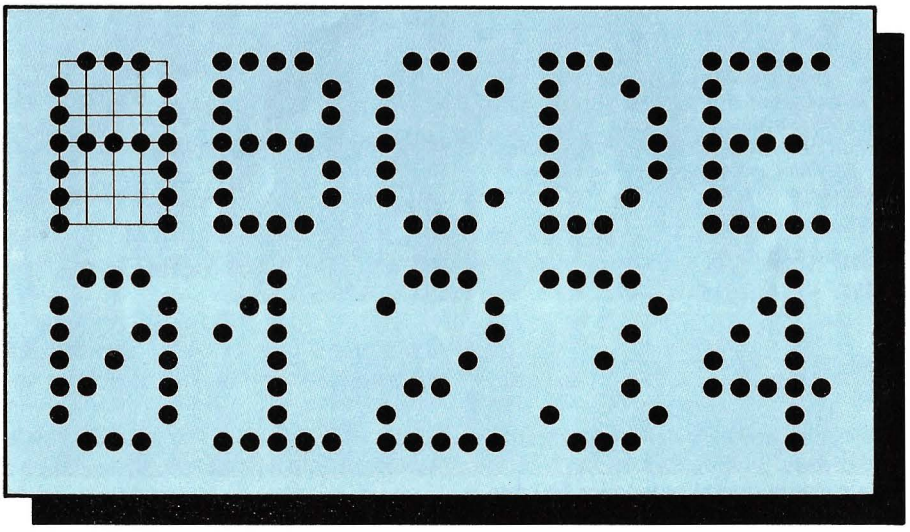
LIST "P"

Obtiene un listado del programa por impresora; permite especificar los números de líneas a imprimir.

FORMATO: LIST "P", {<nl1> – <nl2>}
EJEMPLOS:

```
10 LIST "P"
20 LIST "P", 10 – 1120
30 LIST "P", – 100
```

Algunos fabricantes ofrecen para sus equipos una alternativa que combina la función de impresora con la de plotter o trazador gráfico. Este tipo de periféricos de escritura dibuja los caracteres y las líneas del trazado por medio de "plumas" o bolígrafos de diversos colores.



Composición típica de algunos creados por una impresora matricial de 7 × 5 puntos.

```
40 IF INKEY$="" THEN GOTO 40
50 PRINT "ESA TECLA NO VALE"
```

Así, pues, si el programa es extenso y se está alimentando a la impresora mediante hojas sueltas, se podrá hacer un cálculo del número de líneas de programa que caben en una hoja y, de esta forma, fraccionar el listado en varias hojas. De no hacerlo así, lo más normal es que al llegar al final del papel la impresora siga listando el programa con lo que, al poder avanzar el papel, el resultado sería un inmenso borrón indescifrable en la última línea impresa.

La indicación de los números de línea inicial y final también hace posible obtener listados de sólo algunos trozos interesantes del programa, como subrutinas de uso general o algún segmento especial del mismo.

El lenguaje BASIC no está demasiado estandarizado en este aspecto. Son muchos los intérpretes BASIC que no aceptan el referido comando, siendo necesario el uso de otros comandos alternativos.

Por ejemplo, otro de los comandos frecuentes para el mismo cometido es LLIST, cuyo formato y funcionamiento coinciden con lo indicado en el caso anterior. En otros traductores BASIC, es necesario "abrir" un canal de comunicación entre el ordenador y la impresora; y debiendo indicar este hecho a continuación del comando LIST, por medio de un número de identificación del canal previamente abierto. Este último caso se describirá ampliamente en capítulos sucesivos.

Una vez que se establezca la correspondiente comunicación entre ordenador e impresora, puede ocurrir que el primer listado que se obtenga sea desastroso: con la presencia de líneas en blanco entre cada dos líneas del programa, o sin que se produzca ningún avance del papel, con lo que el resultado será una preciosa línea del color negro más oscuro que contiene, ella sola, todo el listado del programa. Esto puede ser debido a la diferencia existente entre los caracteres de "retorno de carro" y de "salto de línea" que le son enviados a la impresora por el ordenador. En efecto, el ordenador enviará un retorno de carro al final de cada línea de texto que hará retroceder la cabeza de impresión al comienzo de la línea y, tras ello, mandará un carácter de salto de línea que hará avanzar al papel.

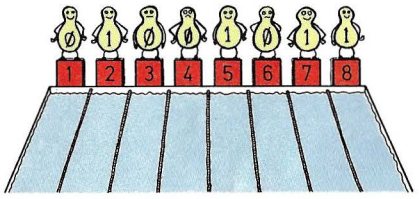
El carácter de salto de línea es generado automáticamente por el ordenador, o por

la misma impresora, con lo que en el caso de concurrir ambos o ninguno de los dos, pueden llegar a producirse los desastrosos resultados descritos. La solución estriba en una lectura cuidadosa de los manuales de usuario, tanto de la impresora como del ordenador en cada caso específico.
Una vez que ya sabe obtener listados por

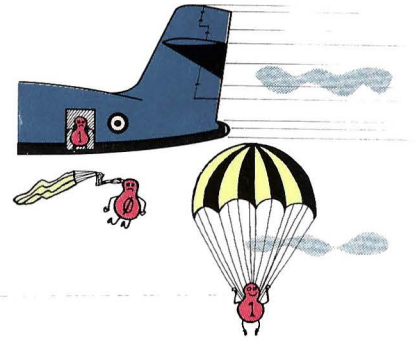
Medios de comunicación

Los dispositivos periféricos que rodean al ordenador y permiten su comunicación con el mundo exterior, exigen la presencia de un medio especializado en el diálogo de entrada o salida, medio que debe facilitar el trasvase de las órdenes de trabajo y de la propia información.
Los dispositivos que permiten establecer tal comunicación obedecen al apelativo genérico de "interfaces". Esta palabra engloba tanto a los circuitos electrónicos que lo integran como al software que lo gobierna.
Internamente, los ordenadores poseen los denominados "buses", formados por varios conductores en paralelo que transportan información binaria. Dicha información corresponde a los datos en proceso, a órdenes de control y a direcciones de memoria. Mediante estas señales, la CPU "sabe" cuándo debe enviar cierta información a un periférico o recibirla de éste, y también conoce si un periférico determinado se encuentra disponible o no. El conocimiento de estas situaciones se obtiene en base al adecuado intercambio de señales de control; tal intercambio se canaliza a través de los interfaces que unen al ordenador con los periféricos, y respetando un determinado protocolo de comunicación previamente establecido.
Normalmente, la comunicación entre el ordenador y la mayoría de los periféricos se establece en el llamado "formato paralelo": a través de un conjunto de cables cada uno de los cuales transporta una señal binaria. Por extensión, los interfaces en los que se apoya este tipo de comunicación se denominan "interfaces en paralelo", y realizan el trasvase de información byte a byte. El tipo de interface más conocido y utilizado en el terreno de la comunicación con impresoras en formato paralelo es el llamado "CENTRONICS"; en razón a la firma que lo creó.
Es indudable que además de las líneas de datos son necesarias toda una serie de señales de control. En el caso del interface CENTRONICS, las señales presentes en el conector de comunicación son las siguientes:

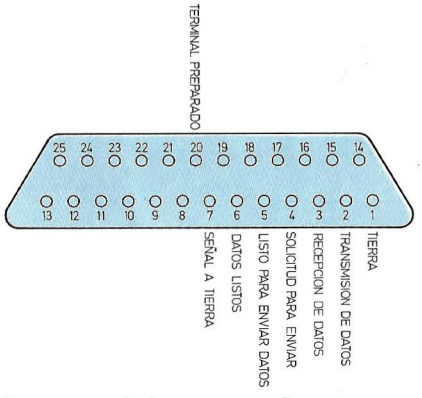
ADK: disposición para aceptar datos.
GND: masa de referencia o "tierra" para el ordenador y periférico.
STROBE: señal del ordenador al periférico que revela que la información está ya preparada para ser leída.
BUSY: señal de periférico ocupado.



En los sistemas de comunicación en formato paralelo, los ocho bits que conforman un byte parten y se transfieren simultáneamente desde el dispositivo emisor.



Si la comunicación se establece en formato serie, los bits son canalizados ordenadamente, uno tras otro, sobre una misma línea de transmisión.



Conector estándar correspondiente a un sistema de comunicación en formato serie según la norma RS/232C. En el gráfico se indica el cometido de los terminales más relevantes.

D0 a D7: ocho líneas de bit para la transferencia de los datos.
Aunque este tipo de interface es el más utilizado para la comunicación con impresoras, la estandarización del mismo no es perfecta.
Es fácil tropezarse con multitud de conectores diferentes que pretenden realizar el mismo tipo de unión, si bien, el problema no pasa generalmente de encontrar el cable adecuado a cada caso.
Cuando los dispositivos a conectar se encuentran a considerable distancia física y su bus de datos incluye un gran número de señales, la conexión en paralelo no es recomendable. En tal caso se opta por la comunicación en "formato serie".
En la comunicación en "serie", se van enviando por un único cable, y uno detrás de otro, los bits que constituyen la información a transmitir. Aunque no se den las condiciones de lejanía antes indicadas, es muy frecuente encontrar ordenadores que disponen de "interfaces de tipo serie" para la comunicación con los dispositivos periféricos.
El tipo de interface serie más frecuente es el que responde a las siglas RS/232. Su definición a nivel teórico está bastante estandarizada por lo que respecta a los niveles de las señales eléctricas y a la asignación de los terminales de conexión.
Se suele utilizar un conector "miniatura D" de 25 contactos, de los cuales el número 2 se emplea para transmitir los datos del ordenador al periférico, el número 3 para recibir los datos procedentes del periférico y el número 7 como masa de referencia o común de ambas señales.
Los dispositivos de transmisión y de recepción deben ser escrupulosamente ajustados en cuanto a velocidad de transmisión y al formato de la información enviada para que la transferencia se produzca correctamente.
Además de los dos citados, existen otros muchos tipos de interfaces; si bien, suelen ser específicos de un determinado modelo o gama de ordenadores.
Generalmente, éstos sólo permiten la transmisión de datos en un sentido, tal es el caso de los interfaces para joystick, para monitores de televisión, para entradas analógicas con fines de medición, para el control del motor de un magnetófono a casetes...

LPRINT

Imprime el contenido de variables, datos literales o expresiones en general.

FORMATO: LPRINT <exp.> {[;] [,] [<exp.>]}
EJEMPLOS:

```
10 LPRINT NUM;
20 LPRINT "ADIOS"
30 LPRINT 5 * 3
```

LPRINT AT

Lista valores por impresora, permitiendo la tabulación en columnas.

FORMATO: LPRINT AT x,y; <exp.>
EJEMPLOS:

```
10 LPRINT AT 0,0; "HOLA"
20 LPRINT AT 10.0; "ADIOS"
```

impresora, el usuario estará ansioso de conocer la forma de imprimir los mensajes que desee o los resultados de sus propios programas. Pues bien, la forma de lograrlo es tan sencilla como la utilizada para los listados. Como ya se adivina, guarda una estrecha relación con los comandos adecuados para ordenar esa misma función sobre la pantalla del TV.

Mientras que para imprimir caracteres sobre la pantalla se utiliza el comando PRINT, para que la salida se produzca por la impresora el comando indicado suele ser LPRINT, cuyo formato más general es el siguiente:

<nl.> LPRINT <exp.>

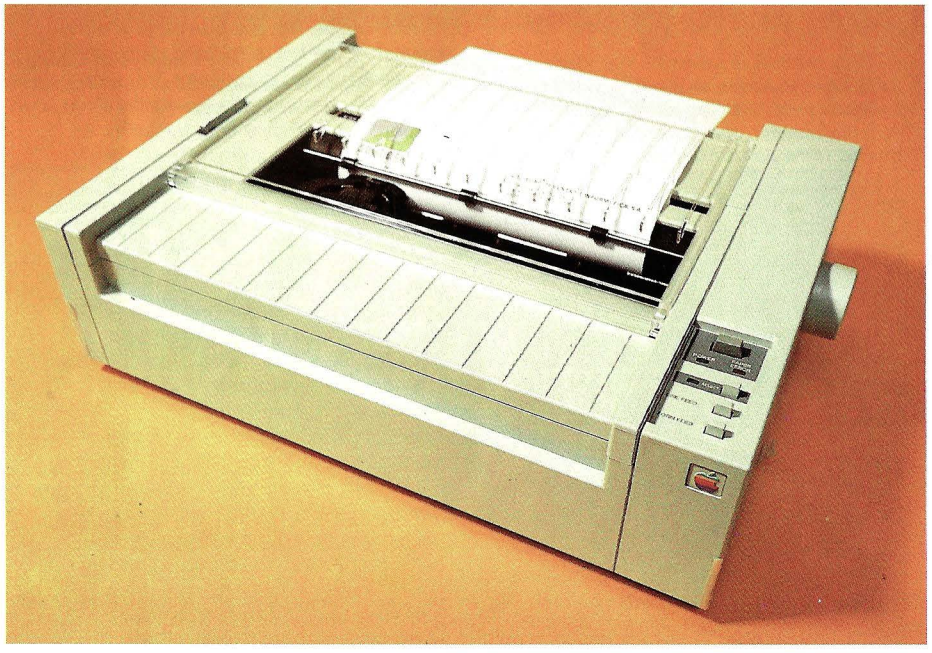
La expresión <exp.> puede ser una cadena de caracteres, una variable, un número directamente o una expresión cuyo resultado se calcula e imprime en papel. Así, por ejemplo, las siguientes líneas:

```
10 LPRINT "HOLA"
20 LET N=1234
30 LPRINT "N=";N
40 LPRINT
50 LPRINT 5678
60 LPRINT 4*6
70 END
■
```

harán que la impresora escriba el texto que sigue:

```
HOLA
N=1234

5678
24
```

Como se puede observar en la línea 30 del ejemplo anterior, es posible utilizar los separadores típicos de la sentencia PRINT, como son la coma y el punto y coma. Ambos producirán los resultados ya cono-



En la fotografía aparece la impresora Imagewriter de la firma Apple. Esta es una impresora de matriz de puntos facultada para la impresión de muy diversos alfabetos (fuentes) de caracteres.

El comando LIST "P" permite obtener listados de programas a través de la impresora en un soporte permanente de papel.

cidos, aunque, en este caso, sobre la hoja de papel.
La orden LPRINT, sin más, produce en la impresora una línea completamente en blanco, ya que en este caso el ordenador únicamente envía al referido periférico los caracteres de retorno de carro y de salto de línea. Por otro lado, es posible utilizar en la mayoría de los casos las características de tabulación propias del comando PRINT AT x,y. Aunque con la salvedad que sólo funcionará el posicionamiento de la cabeza de impresión en la columna indicada, y no será posible su colocación en otra línea del papel distinta a la actual; ello se debe a que, en general, las impresoras no cuentan con mecanismos para el retroceso del papel.
Veamos un nuevo programa a título de ejemplo. Su cometido es escribir aleatoriamente los signos de una imaginaria quiniela futbolística:

```
10 LPRINT "QUINIELA":LPRINT
20 FOR I=1 TO 14
30 LET A=INT(RND(0)*100)
40 IF A>50 THEN LPRINT "1"
50 IF A>15 AND A<=45 THEN LPRINT " X"
60 IF A<=15 THEN LPRINT " 2"
70 NEXT I
```

Partiendo de unas determinadas probabilidades (en porcentaje) y de un número aleatorio, entre 0 y 100, generado por el ordenador, se van obteniendo los diferentes signos: 1, X ó 2. Signos que aparecerán impresos en el papel en su correspondiente columna; un posible resultado es el que aparece a continuación:

```
QUINIELA

1
1
1
  X
1
    2
  X
1
1
  X
  X
1
    2
    2
```

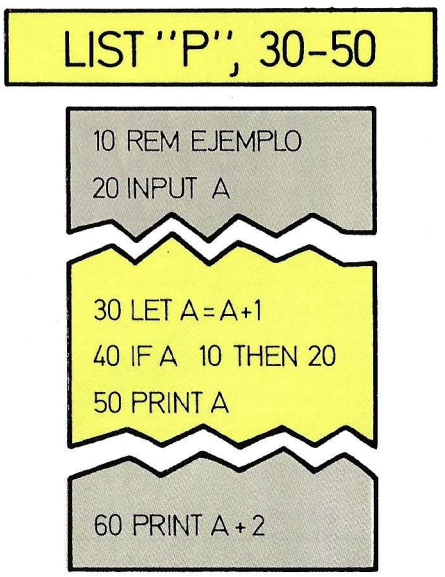
Queda a la atención del lector la posible —y muy sencilla— modificación de este programa para que sea capaz de escribir sobre un boleto original de apuestas en lugar de sobre una hoja de papel convencional.

TABLA DE CONVERSION

ORDENADOR	LLIST	LPRINT
	LLIST [<n1>–<n2>]	**LPRINT** <exp.>
APPLE II (APPLESOFT)	—	—
APRICOT (M-BASIC)	LLIST [<n1>–<n2>]	LPRINT <exp.>
ATARI	LIST "P", [<n1>–<n2>]	LPRINT <exp.>
CBM64	—	—
DRAGON	LLIST [<n1>–<n2>]	PRINT#–2<exp.>
EQUIPOS MSX	LLIST [<n1>–<n2>]	LPRINT <exp.>
HP-150	LLIST [<n1>–<n2>]	LPRINT <exp.>
IBM PC	LLIST [<n1>–<n2>]	LPRINT <exp.>
MPF	—	—
NCR DM-V (MS-BASIC)	LLIST [<n1>–<n2>]	LPRINT <exp.>
NEW BRAIN	—	—
ORIC	LLIST [<n1>–<n2>]	LPRINT <exp.>
SHARP MZ-700 (MZ-BASIC)	LIST/P [<n1>–<n2>]	PRINT/P <exp.>
SINCLAIR QL	—	—
SPECTRAVIDEO	LLIST [<n1>–<n2>]	LPRINT <exp.>
ZX-SPECTRUM	LLIST [<n1>–<n2>]	LPRINT <exp.>

CARACTERES DE CONTROL

Dentro del repertorio de caracteres del código ASCII, empleado normalmente por



Recurriendo a la opción de indicar los números de línea detrás del comando LIST "P", es posible obtener listados parciales del programa.

```
 10 CLS
 20 PRINT "TALONARIO DE CHEQUES"
 30 PRINT
 40 DIM NOMBRE$(20),PESETAS$(50),FECHA$(20),MES$(15)
 50 INPUT "CANTIDAD EN NUMERO";C
 60 INPUT "DESTINATARIO";NOMBRE$
 70 INPUT "CANTIDAD EN LETRA";FECHA$
 80 INPUT "FECHA EN LETRA";FECHA$
 90 INPUT "MES";MES$
100 INPUT "  DE  198";YEAR
110 LET YEAR=YEAR+1980
120 LPRINT "
";CHR$(15);"PTA  #";C;"  pts.#";CHR$(14)
130 LPRINT
140 LPRINT CHR$(15);"Paguese a   ";NOMBRE$
150 LPRINT "Pesetas  ";PESETAS$
160 LPRINT CHR$(14);"                              ";CHR$(15);FECHA$;"
de   ";MES$;"              de ";YEAR;CHR$(14)
180 FOR I=1 TO 5:LPRINT  :NEXT I
```

Listado del programa "Talonarios de cheques".

la gran mayoría de ordenadores para la representación interna de los caracteres, existen una serie de códigos dedicados exclusivamente al control de diversos periféricos y otros menesteres; códigos que no plasman un resultado escrito ni sobre la pantalla ni sobre el papel. Los referidos códigos se utilizan con distinto objetivo en los diversos modelos de impresoras, aunque siempre para activar sus posibles "efectos especiales" de impresión.

Los caracteres de control serán enviados a la impresora, desde el ordenador, de diferentes formas. Los métodos más comunes consisten en utilizar la función CHR$( ) como argumento de un comando LPRINT, o bien mediante la generación de algún carácter alternativo pulsando la tecla CONTROL o ESCAPE y otra simultáneamente.

Entre los "efectos especiales" que se pueden conseguir en la mayoría de las impresoras comerciales, cabe mencionar los siguientes: espaciado proporcional, que consiste en asignar menor espacio en el papel a los caracteres más estrechos como la "i" y más espacio a los más anchos como la "w"; selección del tipo de letra, en las impresoras matriciales entre tipos tales como elite, pica, elite alargada, pica condensada, cursiva, cursiva enfatizada, etc.; doble percusión del martillo, que imprime dos veces superpuestas un mismo carácter, logrando una mejor calidad de letra; activación y desactivación del subrayado del texto a escribir; selección de ciertos caracteres especiales, como la "eñe" española o los signos de diéresis y acentos; modificación de los patrones de caracteres de las impresoras matriciales para así poder crear gráficos y diagramas...

Debido a la casi nula estandarización de los caracteres de control empleados por las impresoras, es posible relatar aquí la función de los mismos. Mientras en un modelo, el código 24 se emplea, por ejemplo, para activar el subrayado, en otro modelo puede no producir efecto alguno, o bien desencadenar una acción completamente distinta.

Para generalizar la cuestión, en el siguiente ejemplo (ver cuadro adjunto), cuya misión es editar cheques impresos, se presupone que el carácter CHR$(15) activa el efecto de subrayado mientras que CHR$(14) lo desactiva. Ambos códigos deben ser modificados en función del modelo de impresora que se utilice.

Un posible resultado de su ejecución es el que aparece en el cuadro adjunto.

Tomando como punto de partida el referido programa, el lector puede practicar el manejo de su impresora y acondicionar el programa para que sea capaz de rellenar cheques auténticos. Aunque, desde luego, la firma siempre debe partir del propio puño y letra del titular de la cuenta.

```
                                        PTS  #50.000 pts.#

Paguese a JOSE PEREZ PEREZ

Pesetas  CINCUENTA MIL

                                TREINTA      de MAYO        de 1985
```

Un posible resultado impreso derivado de la ejecución del programa "Talonario de cheques".

Pascal (y 11)

El último paseo por el PASCAL

La importancia e indudable utilidad del tipo registro (RECORD), estudiado en el capítulo anterior, da pie a describir una serie de conceptos relacionados más o menos directamente con el mismo. Por ejemplo, la forma en que se aprovecha la memoria interna que brinda el ordenador no es un tema relacionado directamente con lenguajes de alto nivel, si bien, cobra cierta importancia cuando éstos son capaces de influir en ella.

El elemento básico de información de cara a su tratamiento por parte del ordenador y a su almacenamiento en memoria, es la palabra binaria. Cada una de estas palabras representará a un dato de tipo simple (número, carácter, valor lógico...) y ocupará una de las posiciones o celdas de la memoria interna. Como quiera que lo habitual es trabajar con palabras binarias de 8 bits, cada una de dichas celdas estará integrada por ocho casillas o elementos de memoria. Esta forma de almacenamiento se denomina desempaquetada (UNPACKED), y constituye el método más natural de guardar la información.

Cada dato ocupará, en consecuencia, una posición de memoria. Sin embargo, si en una posición de memoria (integrada por ocho celdillas de bit) es posible almacenar, por ejemplo, datos numéricos de hasta cinco cifras, no cabe duda que al almacenar un dato con tan sólo dos cifras se estará desaprovechando el espacio de memoria. Si esa información se pudiese "empaquetar" de alguna forma, aumentaría el rendimiento de la máquina.

El PASCAL permite dicho empaquetado de la información mediante el atributo PACKED, aplicable a las estructuras RECORD y ARRAY:

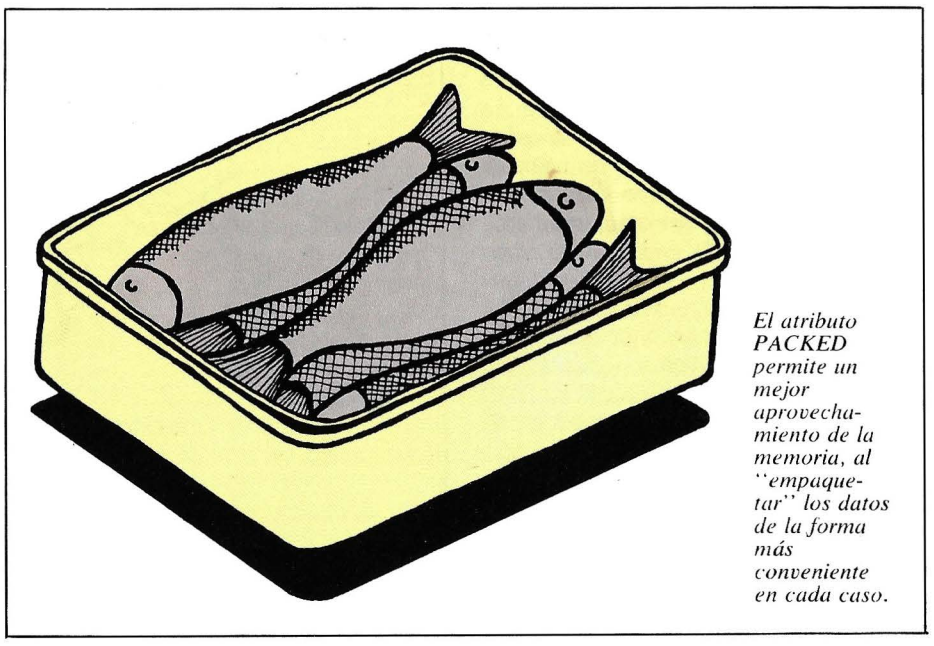
```
TYPE fecha=PACKED RECORD
       dia:1..31;
       mes:1..12;
       año:0..2000
     END;

     tira=PACKED ARRAY [1..15]
     OF CHAR;
VAR hoy:fecha;
    nombre:tira;
```

Así, por ejemplo, en la variable "hoy" el número correspondiente al día se empaquetará junto con el número del mes en una sola palabra. A su vez, en otra palabra se guardará el número del año. De esta forma tan sólo son necesarias dos palabras, mientras que si el almacenamiento se realizara en modo desempaquetado se necesitarían tres palabras: una para cada uno de los números correspondientes al día, al mes y al año. Para registros con campos más complicados, el compilador de PASCAL se encarga de realizar el empaquetamiento de los mismos de la forma más eficaz.

CADENAS DE CARACTERES (STRINGS)

Un tipo de dato utilizado con harta frecuencia en los programas es el "string" o "cadena de caracteres". Este se define como una matriz empaquetada de caracteres de la forma que se indica en el ejemplo anterior. Su uso permitirá manejar una secuencia de caracteres que representen, por ejemplo, nombres de personas u objetos, ya que como se recordará las variables de tipo CHAR sólo podían almacenar un único carácter.

Una de las operaciones más frecuentes con este tipo de datos es la de clasificarlos por orden alfabético; para ello se pue-



El atributo PACKED permite un mejor aprovechamiento de la memoria, al "empaquetar" los datos de la forma más conveniente en cada caso.

den utilizar los operadores lógicos de relación: <,>,=,< >, <=,>=... Una ventaja de las cadenas de caracteres es que permiten asignar un valor a todos sus elementos simultáneamente, obviando la necesidad de realizar esta asignación carácter por carácter. Una limitación al respecto es que el número de caracteres debe coincidir con el tamaño declarado para la cadena en cuestión. Si el número de caracteres que interviene en la asignación no coincide con el tamaño declarado para la cadena, no será posible realizar la asignación desde el exterior mediante una sentencia de tipo READ. En tal caso, será necesario hacerlo carácter a carácter, por ejemplo, mediante bucles FOR. No obstante, sí será posible proyectar una cadena completa al exterior mediante la correspondiente sentencia WRITE, siendo el tamaño del campo del formato de salida igual al tamaño de la cadena de caracteres a extraer.

REGISTROS VARIANTES

La estructura de las variables ordinarias de tipo registro es bastante rígida; de tal forma que todos los registros de un mismo tipo tendrán el mismo número de campos, con los mismos nombres y con los mismos tipos.
En muchos casos puede ser conveniente hacer que esta estructura resulte un poco más moldeable, con objeto de adecuarla a las posibles variaciones de los datos en cuanto al número y tipo de los campos. Esto se consigue mediante los llamados registros variantes, cuyo método de definición es el siguiente:

```
TYPE <identificador>=RECORD
     <campo1>:<tipo>;
     <campo2>:<tipo>;
     ...
     CASE <campo distintivo>:
          <tipo> OF
          <valor1>:([id.1>:<tipo>;
          <id.2>:<tipo>;...)
          <valor2>:([id.1>:<tipo>;
          <id.2>:<tipo>;...)
          ...
     END;
```

Por ejemplo:

```
TYPE datos=RECORD
     nombre, apel1, apel2:PACKED
     ARRAY [1..20] OF CHAR;
     nacim:fecha;
     nacionalidad:(ESPANYOLA,
     EXTRANJERA);
     CASE civil:(SOLTERO,
          CASADO) OF
          SOLTERO:(novia:BOLEAN);
          CASADO:(boda:fecha;mujer:
          ARRAY[1..20] OF CHAR)


     END;
```

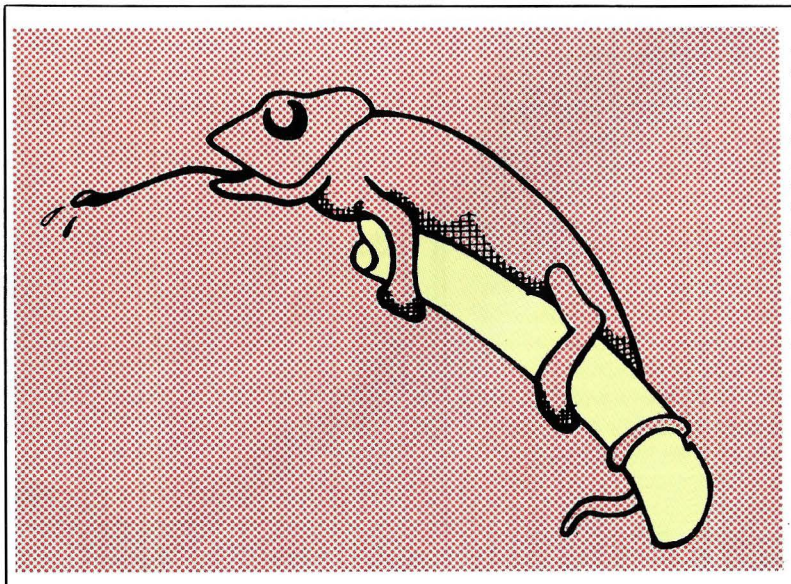
El denominado "campo distintivo" actúa como seleccionador de la sentencia CASE, la cual determina el número y tipo de campos requeridos en cada circunstancia según su valor actual; ello proporciona una mayor flexibilidad si cabe a los datos de tipo RECORD.

PUNTEROS

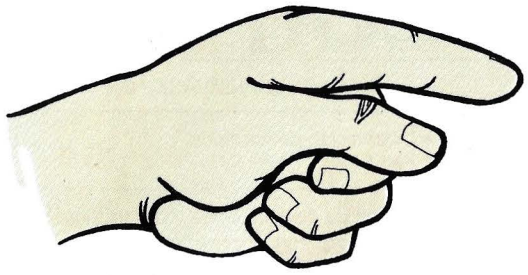
El lenguaje PASCAL posee un tipo de variables denominadas "punteros" o "apuntadores" (POINTERS), cuya misión es almacenar la dirección de memoria de un determinado dato. Con su colaboración es posible enlazar los datos de forma que la cantidad de memoria ocupada por un determinado conjunto de datos sea únicamente la imprescindible, ni más ni menos. Esta característica se conoce como estructura dinámica de datos. Existen procedimientos que permiten enlazar unos datos con otros, borrarlos o adiccionarlos; y todo ello sin necesidad de volver a reorganizar toda la estructura ni definir previamente el tamaño necesario, como ocurriría con las variables de tipo ARRAY, en el caso que se pretendiera utilizarlas para

```
VAR NOMBRE = ARRAY [1.....20] OF CHAR;
NOMBRE := 'FRANCISCO'
```

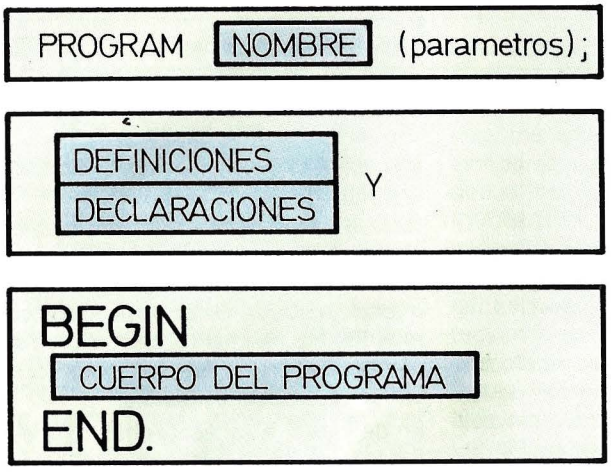
Mediante la definición de un tipo ARRAY de caracteres, es posible manejar con comodidad y eficacia las denominadas cadenas de caracteres o "strings"



Los registros variantes acomodan el número y el tipo de sus campos a las circunstancias actuales; al igual que un camaleón acomoda su color al del terreno que lo circunda.



Los punteros señalan a una posición de memoria que contiene un dato de interés para el programa.



La estructura de un programa PASCAL resulta muy sencilla de aprender y fácil de poner en práctica; lo mismo ocurre con la estructura de sus elementos integrantes: procedimientos, funciones...

producir un número de elementos no determinado de antemano.
Las variables de tipo puntero se definen de la siguiente forma:

TYPE <identificador>: ^ <tipo>;

siendo el signo "^" el que diferencia a este tipo de dato, y <tipo> el parámetro que señala el tipo de datos a los que apuntarán estas variables (puede ser cualquier tipo incluso estructurado).
Para inicializar la variable puntero y reservar así espacio de memoria para la primera variable, se dispone del procedimiento: NEW (<variable apuntador>);
La operación contraria corre a cargo del procedimiento: DISPOSE (<variable apuntador>);
Para acceder a dicha posición reservada se hace uso de la notación siguiente: <variable apuntador> ^ creada al ejecutar un NEW; para pasar a la posición siguiente se ejecutará un nuevo NEW. Si se desea que la variable puntero no señale a ningún elemento, habrá que asignar a ésta el valor NIL.
Mediante la definición de los registros adecuados con algún campo de tipo puntero, será posible crear listas encadenadas de datos con carácter dinámico que, evidentemente, potenciarán el rendimiento del ordenador.

PASCAL: EL LENGUAJE DE LOS 80

El PASCAL es un lenguaje de programación muy atractivo y elaborado. Actualmente cuenta con una gran difusión, sobre todo en las esferas de la enseñanza, debido a que en un principio nació como un lenguaje con fines educativos. Por esta misma razón presenta ciertas limitaciones que lo alejan de ser un lenguaje idóneo para ciertas categorías de aplicaciones muy específicas. La filosofía del PASCAL se resume en el objetivo de conseguir que la programación se convierta en una tarea muy sencilla y sistemática, de forma que la creación de programas se pueda realizar poniendo en práctica métodos muy sencillos de comprender y recordar. Es por ello que el repertorio de instrucciones del PASCAL no es tan exhaustivo como puede ser el de otros lenguajes, como el BASIC o el LOGO; no obstante, estos últimos oponen a su naturaleza de lenguajes aptos para un mayor número de aplicaciones, una superior complejidad en la codificación de los programas. A pesar de todo, el PASCAL cuenta en nuestros días con una gran aceptación, incluso en el marco de la creación de software profesional.
La principal característica, y a la vez el mayor éxito del PASCAL, reside en su naturaleza de lenguaje estructurado, tanto por lo que respecta al programa como a los datos. Esto hace que la legibilidad y flexibilidad de modificación de los programas en PASCAL sea, con mucha diferencia, superior a la de otros lenguajes no estructurados. También contribuye a esta sencillez y claridad el formato libre de escritura de los programas. Así, con un simple vistazo será posible comprender fácilmente la misión encomendada al programa, y conocer qué funciones realizan cada una de sus zonas; no hay que olvidar que otra de las características importantes del PASCAL es su modularidad, esto es: un programa en PASCAL se puede dividir en pequeños módulos o segmentos más fáciles de manejar y comprender.
También se puede considerar como una peculiaridad el hecho de que un programa fuente escrito en PASCAL siempre debe ser traducido por medio de un compilador, y no a través de un intérprete. La compilación creará el denominado programa objeto, escrito en lenguaje máquina, que podrá ser ejecutado directamente siempre que no evidencie ningún tipo de error.
El hecho de tener que someterse a un proceso de compilación hace que resulte obligada la declaración de las variables y demás elementos que vayan a ser utilizados posteriormente en el programa; de esta forma, el compilador reservará el espacio de memoria necesario para dichas variables, lo que garantizará el correcto funcionamiento del programa.
Las últimas tendencias en las técnicas de programación se ven reflejadas en el PASCAL; un lenguaje relativamente moderno. Así, por ejemplo, se presta una gran atención a la estructura de los datos y a su definición abstracta; como se señaló al principio, ello contribuye a la legibilidad del programa y facilita la explotación de la componente dinámica de éstos. La manipulación de los datos se ve favorecida, además, por su diferenciación en "tipos",

TABLA DE COMANDOS-PASCAL

COMANDO	COMETIDO	OBSERVACIONES
TYPE <id.>=PACKED <tipo>;	Empaquetamiento de los datos para optimización de la memoria.	Definición de los tipos: RECORD o ARRAY.
TYPE <id.>=RECORD...CASE<dist.>OF<lista>;	Definición de registro variante.	Zona de declaraciones y definiciones.
TYPE <id.>=^<tipo>;	Definición del tipo puntero.	Zona de declaraciones y definiciones.
NEW (<punt.>);	Inicialización de un puntero.	Procedimiento estándar.
DISPOSE (<punt.>);	Eliminación de un puntero.	Procedimiento estándar.

NOTA: <id.>: identificador válido PASCAL. <tipo>: tipo de dato. <dist.>: campo distintivo como selector del CASE. <lista>: lista de los valores del CASE que indiquen los campos en cada caso. <punt.>: variable de tipo puntero.

lo que permite la detección de errores difíciles de descubrir de otro modo.

Los tipos de datos, clasificables en estructurados y no estructurados, se definen mediante la declaración de tipo TYPE. En PASCAL, es posible utilizar cuatro tipos de datos escalares estándar: REAL, INTEGER, BOOLEAN y CHAR, todos ellos no estructurados. Los tres primeros tipos permiten la manipulación de datos numéricos y lógicos de la forma convencional, mientras que el tipo CHAR está destinado al manejo de caracteres alfabéticos, numéricos y signos especiales.

Además de estos tipos, siempre disponibles en cualquier compilador, el PASCAL permite al usuario la definición de sus propios tipos de datos, según dos procedimientos: por enumeración de los posibles valores que podrá adoptar una variable de tal tipo, y como subrango o subconjunto de los valores de otro tipo previamente definido.

La posibilidad de estructuración de los datos es la que otorga una nueva dimensión a este lenguaje. Así, cuenta con los siguientes tipos estructurados: ARRAY, SET, RECORD y FILE, además de con la posibilidad de crear estructuras dinámicas mediante el empleo de las variables de tipo puntero.

La declaración de las variables se efectúa siempre después de la definición de los tipos, y mediante la palabra reservada VAR. Las constantes tienen también un tratamiento especial en PASCAL: se pueden declarar mediante la palabra CONST lo que permitirá hacer referencia a una constante mediante un identificador. Esto último aporta la ventaja de que si dicho valor constante se quiere modificar en un momento dado, sólo habrá que modificar el valor asignado a su identificador, sin tener que tocar el resto del programa; de nuevo sale a relucir el carácter sistemático y clarificador del PASCAL.

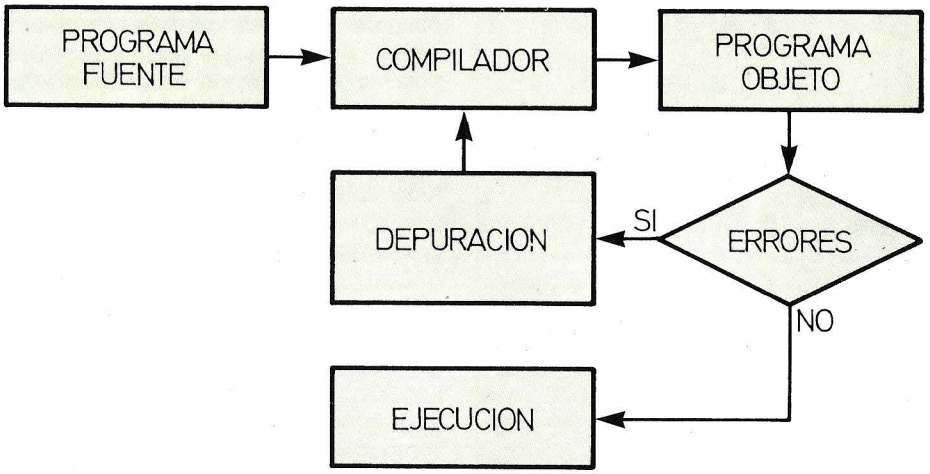
Todos los elementos del PASCAL mantienen la misma estructura: una cabecera, una zona de declaraciones, y, finalmente, el cuerpo principal de sentencias.

Incluso un programa completo mantiene esta misma estructura, de tal forma que cabe considerarlo como un superprocedimiento. A su vez, cada uno de los procedimientos es, realmente, un miniprograma con entidad propia. Las funciones conservan también esta estructura.

En definitiva, la cabecera determina la naturaleza de cada elemento, además de otorgarle un identificador y detallar los parámetros necesarios para la comunicación del mismo con el resto del programa, e incluso con el exterior. La zona de declaraciones se encarga de compendiar todos los elementos que se utilizarán posteriormente en el cuerpo principal del programa; esto irá encerrado entre las palabras BEGIN y END. Las referidas palabras se utilizan también para englobar sentencias simples en una única sentencia denominada compuesta; de ahí que también pueda considerarse al propio cuerpo del programa como una sentencia única.

Todo el "peso" del programa recae en la zona de declaraciones, en la que se definen los procedimientos y funciones; éstos son los principales elementos para la ejecución de acciones, hasta el punto que la mayor parte de las instrucciones del PASCAL son procedimientos o funciones estándar (READ, RESET, NEW, WRITE, INT, SIN...).

Realmente, la programación en PASCAL no resulta tediosa ni aburrida, y mucho menos a partir del momento en el que se ha llegado a una cierta familiaridad con la práctica. Su empleo crea además hábitos de programación muy recomendables, eficaces y aplicables a otros ámbitos de la informática.



El proceso de compilación de un programa puede ser lento y tediosa su depuración; no obstante, la ejecución del programa objeto (compilado) resulta muy rápida.

Apple DOS 3.3 (4)

Métodos de almacenamiento en disco

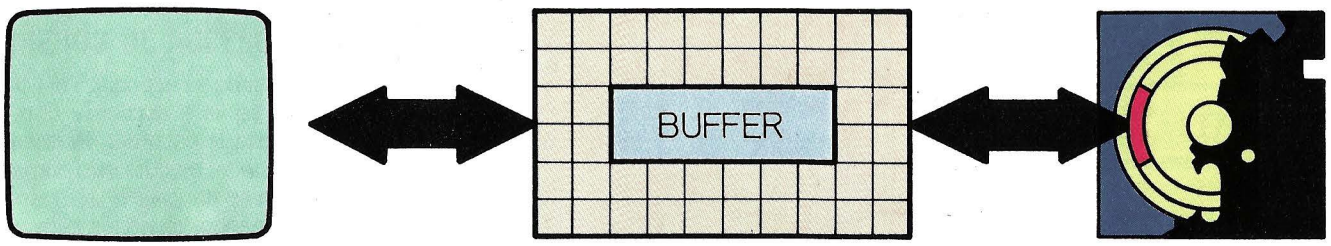
Aunque los equipos Apple II, Apple II plus y Apple IIe resultan muy adecuados para actuar en un ambiente doméstico, el fabricante no ha descuidado en absoluto el manejo de información por medio de ficheros. Las facultades del DOS 3.3 para la gestión de ficheros otorga a estos equipos unas elevadas prestaciones por lo que respecta al almacenamiento masivo de datos; facultades restringidas en cierto grado al uso de unidades de disco flexible, debido a que el sistema operativo trata a los discos rígidos como una zona de almacenamiento de datos dividida en múltiples disquetes "lógicos" (un disco rígido de 5 Megabytes quedaría dividido en 40 ó 50 disquetes "lógicos"). Realmente, ello limita el tamaño máximo de los ficheros, ya que no es posible exceder del espacio propio de un disquete; por lo demás, se desaprovechan varias zonas del disco rígido que quedan reservadas para la definición y localización de los distintos disquetes "lógicos".

ESTRUCTURA FISICA

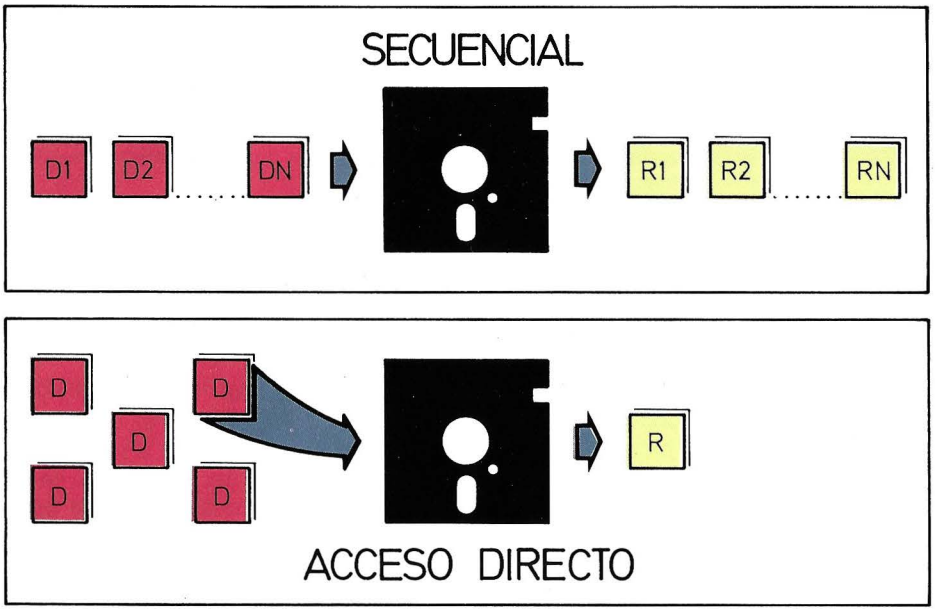
La información contenida en los ficheros residentes en disco flexible se graba sobre la superficie magnética de este en forma binaria (ceros y unos). Ello hace necesaria la organización de la superficie del disquete en pequeñas parcelas, subdividiendo el espacio disponible de manera que la localización de un elemento de información se realice de una forma lógica y ordenada. Esta subdivisión se puede comparar con la que existe en un país (disquete), que generalmente se desglosa en provincias (pistas), las provincias en ciudades (sectores), éstas, a su vez, en calles (bytes) y, por último, cada calle en inmuebles con sus respectivos números (bits).
Los disquetes gobernados por el sistema operativo DOS 3.3 de Apple están divididos en 36 bandas concéntricas o pistas, conteniendo cada una de ellas 16 sectores de 256 bytes cada uno. También cabe hablar del formato de 13 sectores por pista, utilizado en las revisiones del sistema operativo precedentes a la 3.3 (3.1 y 3.2). Así pues, no son intercambiables los disquetes de las distintas revisiones; a no ser que se utilice un programa especial, proporcionado por Apple, adecuado para la conversión del formato de 13 sectores/pista al de 16 sectores/pista.
El proceso de grabación en un disquete no se produce necesariamente cada vez que se ejecuta una sentencia de escritura sobre dicho disquete. Realmente, se utiliza una zona de memoria intermedia o "buffer", en donde se van almacenando las distintas informaciones a grabar hasta que la capacidad del "buffer" (256 bytes) se ve completada. En ese instante se transfiere el contenido de la memoria intermedia al primer sector libre que se encuentre en el disco. Este proceso continúa hasta que se haya producido la grabación completa del fichero.

Parte del espacio del disquete se destina a almacenar la información necesaria para que el sistema operativo sepa, en cada instante, cuáles son los sectores que corresponden a cada fichero, cuántos ficheros hay en el disquete, cuáles son sus nombres, tipo, longitud, etc.

Este espacio no disponible para el usuario está localizado en las pistas 0, 1, 2 y 17. Las tres primeras se reservan para la lista de pistas y sectores utilizados, con el fin de conocer en todo momento la situación de los sectores de los diferentes ficheros. A su vez, la pista 17 es ocupada por el directorio del disquete; en él se reflejan el nombre de los ficheros, sus respectivos tipos y número de sectores que utilizan, la indicación de si están bloqueados o no, además de la localización del primer sector de la lista de pistas y sectores asociados a cada fichero. Compartiendo la pista 17 con el directorio, se encuentra la "tabla de contenidos del volumen" (Volume Table of Contents) VTOC; en ella se especifica el estado, libre u ocupado, de los diversos sectores contenidos en las pistas del disquete.



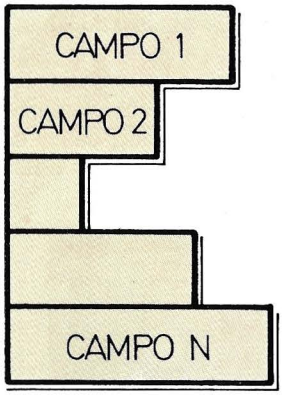
Las operaciones de entrada/salida sobre el disco flexible se efectúan a través de un buffer o memoria intermedia, cuya capacidad iguala a la de un sector del disco. Dichas operaciones no se realizan hasta que se llene por completo el buffer o hasta que el sistema operativo se vea obligado a ejecutarlas.



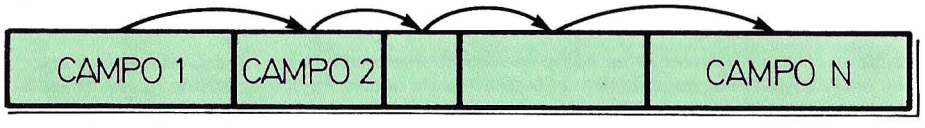
El uso de ficheros secuenciales o de acceso directo depende de las propias características de la tarea a realizar. La correcta elección de uno u otro tipo de estructura supone evaluar algunos factores críticos; por ejemplo: el espacio disponible en disco, la cantidad de datos a procesar y el tipo de tratamiento que deben recibir.

FICHEROS SECUENCIALES

El sistema operativo DOS 3.3 soporta dos modos de operación con ficheros de texto: secuencial y de acceso directo o acceso al azar.
La puesta en práctica de uno u otro modo de operación depende del uso que deba darse a cada fichero y, en definitiva, a la información residente en cada uno de llos.
Los ficheros de acceso secuencial están formados por series de campos, que pueden ser de longitud variable, y cuyo punto final está señalado por la presencia de un carácter especial: el de retorno de carro (carriage-return).
El acceso a los distintos campos se produce de forma secuencial; esto es: se empieza por el primer campo del fichero, a continuación el segundo, y así campo tras campo, ordenadamente, hasta el final del fichero; recorriéndolos en el mismo orden en el que están almacenados.
Los diversos modos de acceso, en lectura o escritura, y el propio acceso al disco se definen por medio de comandos del sistema operativo. La acción de leer o escribir se ordena con las sentencias básicas destinadas a la entrada de datos por teclado y salida por pantalla; la entrada y salida es redireccionable con la intervención de los comandos que a continuación se describen.



Los ficheros secuenciales están constituidos por campos de longitud variable que han de ser accedidos, necesariamente, unos a continuación de otros y en estricto orden.



- OPEN

Naturalmente, un fichero no es accesible si previamente no ha sido abierto. En consecuencia, éste debe de ser el primer comando que se ejecute a la hora de intentar el acceso a un fichero. El resultado de la ejecución del comando OPEN es la creación de un área intermedia de memoria o "file buffer", de 256 bytes de longitud, en la cual se almacena parte del contenido del fichero sobre el que se está actuando. Adicionalmente, el puntero que indica cuál es el próximo campo a leer o escribir, se posiciona al principio del fichero.
El número máximo de ficheros que pueden estar abiertos simultáneamente se eleva a 16, si no se ha especificado un número menor por medio del comando MAXFILES.
La sintaxis del comando OPEN es la siguiente:

OPEN nf [,Sn] [,Dn] [,Vn]

nf: nombre del fichero que va a ser abierto. Si no existe es creado automáticamente.
Sn: número de conector (1 a 7) que contiene la tarjeta que controla el acceso a la unidad de disco.
Dn: indica cuál de las dos unidades de disco controladas por una misma tarjeta contiene al fichero especificado; "n" puede ser 1 ó 2.
Vn: número del volumen del disco que va a ser accedido (de 0 a 254).
Nota: Los corchetes indican que su contenido es opcional.

- CLOSE

Al incluir el trabajo sobre un fichero, éste debe ser cerrado; ello asegurará que todos los caracteres escritos en el fichero son los correctos y, a su vez, quedará liberada el área de memoria intermedia reservada para el fichero.
Sintaxis:

CLOSE [nf]

Si no se especifica el nombre del fichero, serán cerrados todos los ficheros que se encuentren abiertos en ese instante.

- WRITE

Con este comando se especifica el fichero sobre el que se va a escribir y el punto donde comienza la escritura. Permanece activo hasta que se ejecute otro comando del DOS o una sentencia INPUT. Tras ejecutar este comando, todos los caracteres que saldrían normalmente por pantalla son dirigidos hacia el fichero que se indique.

WRITE nf [,Bn]

Bn: número de bytes del campo que han de saltarse antes de empezar a escribir en el fichero.

- READ

Señala el fichero que va a recibir las próximas operaciones de lectura. El sistema operativo leerá del fichero especificado hasta que se ejecute un nuevo comando.
Sintaxis:

READ nf [,Bn]

- POSITION

Permite el acceso a cualquier campo de un fichero secuencial. Cuando se ejecuta, el fichero especificado es abierto automáticamente y el puntero se posiciona tantos campos más abajo del comienzo del fichero como se indique en el argumento del comando.
Sintaxis:

POSITION nf [,Rn]

Rn: el parámetro n indica el número del campo del fichero en el cual se desea posicionar el puntero. En el caso de que no se especifique este argumento, el puntero señalará al primer campo.

- APPEND

Abre un fichero de acceso secuencial y posiciona el puntero detrás del último campo ya utilizado. De esta forma, la nueva información se añadirá a continuación de los campos ya existentes.

APPEND nf [,Sn] [,Dn] [,Vn]

Una vez preparado el acceso al fichero mediante los comandos anteriores, las operaciones de lectura y escritura se llevan a cabo con las sentencias BASIC: PRINT, INPUT y GET. PRINT sirve para escribir el contenido de una variable en el fichero. INPUT para leer el contenido de un campo del fichero hasta la primera coma. Mientras que GET lee todo el contenido de un campo, carácter a carácter.

FICHEROS DE ACCESO DIRECTO

Los ficheros de acceso directo tienen la particularidad de que no es necesario re-

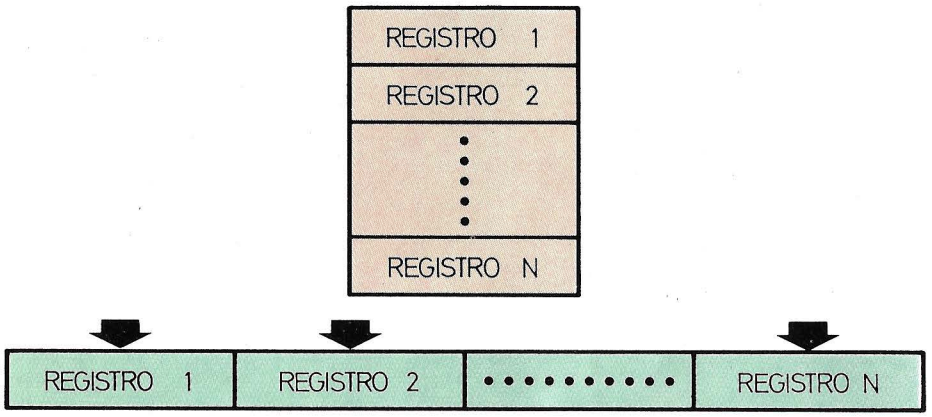
La importancia que adquiere el tratamiento de ficheros viene avalada por la gran cantidad de programas de aplicación que los utilizan profusamente para manipular la más diversa información.

Formatos en pantalla

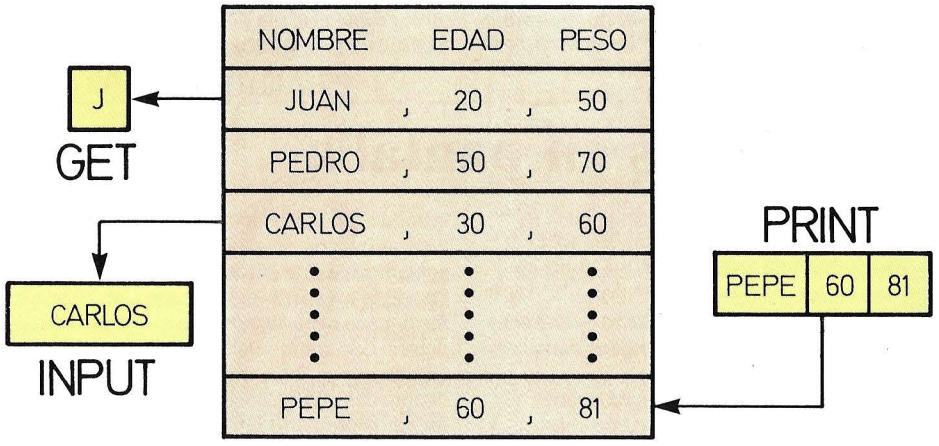
Uno de los aspectos fundamentales de toda actividad informática reside en el modo en que intera cionan el ordenador y el usuario a nivel sensorial. De su importancia nos habla el hecho de que este aspecto condiciona los métodos de trabajo del usuario.
Normalmente, además del ordenador, el usuario emplea material de tipo impreso; y de cara a una interacción más efectiva es conveniente que la forma de introducir los datos en el ordenador adopte una distribución similar a la de dicho material. Es en este punto donde aparecen los formatos de pantalla.
En su acepción más amplia, un formato de pantalla es un procedimiento para la introducción y presentación de datos en el ordenador, con un aspecto y estructura adecuados a una situación específica.
Desde la perspectiva del usuario, el formato de pantalla toma el aspecto de una ficha o documento de identificación, en el que aparecen libres una serie de espacios destinados a la introducción o recuperación de datos por parte del usuario.
Internamente, el formato de pantalla es controlado por un software especial que interacciona con el programa de aplicación de forma transparente para el usuario, y por medio del cual es posible controlar la presentación de los datos en todos o en parte de los campos del formato.
La característica fundamental del formato de pantalla, por lo que respecta a su operación, es que ésta se realiza en lo que se denomina "modo bloque". Ello significa que el conjunto de datos que contiene la pantalla de formato se transfiere de una sola vez, en bloque. Cosa que no sucede en el transcurso de un programa normal, en el cual serían necesarios varias operaciones de entrada y salida por pantalla para su transmisión desde el programa al usuario, o viceversa. Al operar con un formato, la transferencia en bloque se realiza al introducir la orden al efecto (normalmente al pulsar la tecla <enter>).

correr todos los registros que proceden al registro que se desea acceder; para poder llegar hasta el mismo, es suficiente con indicar cuál es el lugar que ocupa dentro del fichero.

La estructura de este tipo de ficheros está integrada por registros de longitud fija que, a su vez, pueden estar divididos en varios campos. Así, un fichero de acceso directo puede compararse con un libro, dividido en hojas (registros) que contiene, cada una, un determinado número de líneas (campos). Los comandos que permiten la explotación de este tipo de ficheros son muy parecidos a los descritos anteriormente para los ficheros secuenciales. Tan sólo varían en el hecho de que en los ficheros de acceso directo es necesario especificar la longitud del registro y el número de registro a seleccionar. A continuación se describen los comandos para ficheros de acceso directo.



Los ficheros de acceso directo están constituidos por registros de longitud fija, accediéndose independientemente a cada uno de ellos, y sin necesidad de pasar por todos los registros anteriores para llegar a uno específico.



Con las sentencias PRINT, INPUT y GET que brinda el lenguaje BASIC, se ve facilitado el acceso a la información contenida en los distintos ficheros del sistema.

- OPEN

Abre un fichero permitiendo el acceso al mismo. El resultado es análogo al que se obtiene en el caso de ficheros secuenciales, exceptuando lo relativo al puntero indicador del campo, ya que este sólo tiene sentido en los ficheros secuenciales.
Sintaxis:

OPEN nf [,Ln] [,Sn] [,Dn] [,Vn]

Ln: n expresa la longitud de los registros que componen el fichero (1 a 32767).

- CLOSE

Libera el "buffer" asociado al fichero, cerrando el acceso al fichero en cuestión.
Sintaxis:

CLOSE nf

- WRITE

Identifica el fichero y el número de registro que se verá afectado por las operaciones de escritura.
Sintaxis:

WRITE nf [,Rn] [,Bn]

Rn: n indica el número del registro que recibirá los datos (de 0 a 32767).

- READ

Identifica el fichero y el número del registro a leer.
Sintaxis:

READ nf [,Rn] [,Bn]

LOS FICHEROS EN LA PRACTICA

La decisión de utilizar ficheros secuenciales o de acceso directo no siempre es fácil de adoptar. En ciertos casos resultan más ventajosos los ficheros secuenciales y en otros los de acceso directo. En general, el uso de unos u otros viene determinado por el resultado de sopesar varios factores, haciendo hincapié en los más críticos. Estos factores son:

— Espacio en disco: dado que los ficheros de acceso directo han de tener constante la longitud de registro, el desperdicio de espacio en disco puede llegar a ser considerable si tan sólo se utiliza una pequeña parte del registro en la mayoría de los casos. En el modo de acceso secuencial, al ser los campos de longitud variable el espacio es aprovechado con mayor eficacia.

— Cantidad de datos: si la mayor parte de los datos contenidos en un fichero van a ser tratados por un programa, lo más razonable es leerlos de una forma secuencial y almacenarlos en memoria para que su acceso sea muy rápido, grabándolos de nuevo en disco una vez que haya terminado su tratamiento.

— Uso de los datos: si el tratamiento de los datos exige que éstos sean accedidos de forma no secuencial, el empleo de ficheros de acceso directo economizará tiempo con respecto a los secuenciales.

Software educativo

Aplicaciones informáticas para la enseñanza

En los planes de enseñanza está haciendo acto de presencia una nueva asignatura que cada vez adquiere una mayor importancia: la Informática. En un futuro próximo es de suponer que no sólo sabrán informática los informáticos, sino que esta disciplina se estudiará como materia auxiliar en otras especialidades. El alto número de alumnos demandantes de educación informática está desbordando la capacidad docente oficial y privada. Esta circunstancia ha permitido el florecimiento de una nueva especialidad dentro de las empresas dedicadas a producir aplicaciones informáticas: el software educativo.

En general, podemos definir como software educativo a aquellos programas de ordenador cuya misión fundamental es educar a los usuarios, ya sea aportándoles nuevos conocimientos (informáticos o de cualquier otro tipo), o bien entrenándolos en materias de cualquier disciplina comúnmente estudiadas por procedimientos tradicionales.

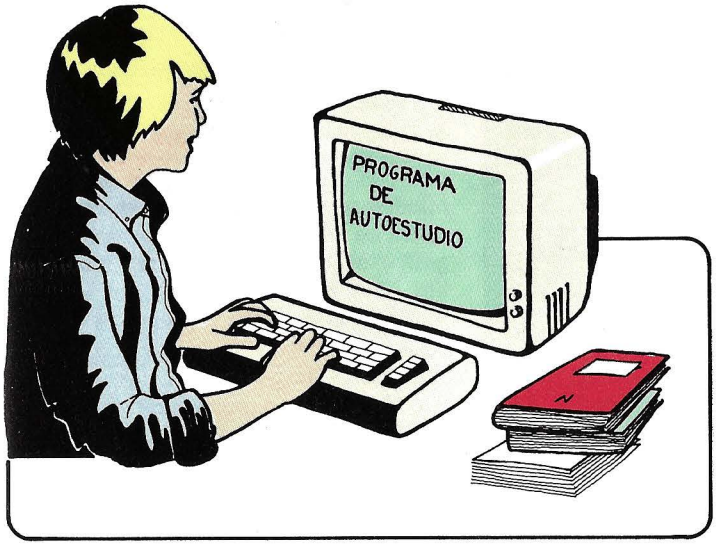
CLASIFICACION DEL SOFTWARE EDUCATIVO

Al igual que en otras especialidades informáticas, dentro del software educativo cabe diferenciar varios grupos de programas con características comunes. A grandes rasgos, podemos hablar de tres tipos de programas educativos: programas de autoestudio, programas de apoyo y programas de educación en informática. A continuación vamos a describir las características fundamentales de los tres grupos citados:

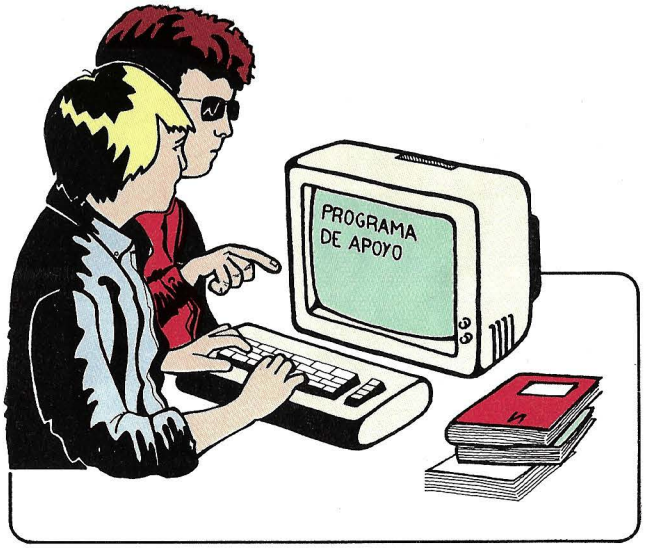
- *Programas de autoestudio*

Denominamos programa de autoestudio a aquellos que, estando dedicados a la educación, permiten al usuario adquirir conocimientos de una determinada materia de forma autónoma. Para conseguir que un paquete educativo pueda ser catalogado como de autoestudio, el "afinamiento" y claridad de la exposición deben ser máximas, ya que el usuario no contará con el apoyo de un profesor que le resuelva las posibles dudas. Por consiguiente, estos programas deben cuidar al máximo no sólo "lo que dicen", sino también "cómo lo dicen". Para suplir de alguna forma las aclaraciones que daría un profesor a su alumno, los programas de autoestudio suelen venir acompañados por una documentación apropiada para el objetivo que se persigue.

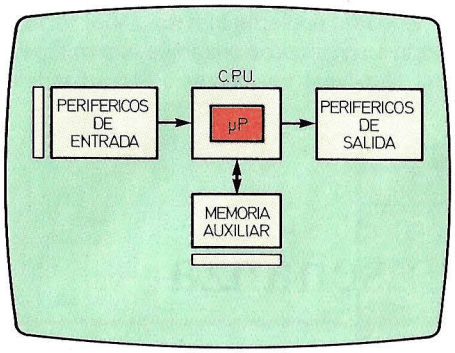
No es posible establecer con precisión una metodología apropiada para el funcionamiento de un programa educativo de



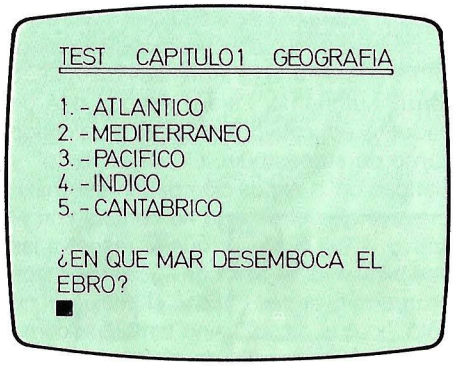
Los programas educativos de autoestudio se caracterizan por la concurrencia exclusiva del alumno y el sistema didáctico (software, ordenador y documentación).



Los programas educativos de apoyo, además de establecer un diálogo directo con el alumno, necesitan la participación de un profesor.



Un grupo importante, dentro del software educativo, lo constituyen los programas destinados a la enseñanza de la Informática.



Una de las fases más importantes de un programa educativo es la proposición de preguntas que permitan al usuario medir sus conocimientos sobre la materia estudiada.

autoestudio. Será la propia materia estudiada y el ordenador disponible, quienes permitirán utilizar unas técnicas u otras; no obstante, podemos citar al menos tres puntos básicos que deben contemplarse en cualquier caso:

1. Introducción.
Con objeto de ofrecer una primera aproximación a la materia estudiada y a la forma en la que ésta se introducirá, el programa debe empezar con una presentación en la que se indique, entre otras cosas, los objetivos a cubrir en el curso, un índice de los capítulos que se estudiarán, el tiempo aproximado necesario para cada capítulo y, en definitiva, cualquier aclaración que pueda resultar interesante para el usuario.

2. Exposición.
Después de la introducción, el programa debe ofrecer al usuario, de forma interactiva, la posibilidad de que elija uno de los capítulos. Una vez seleccionado empezará su exposición, que estará basada, fundamentalmente, en textos resumidos con abundantes cuadros y gráficos.

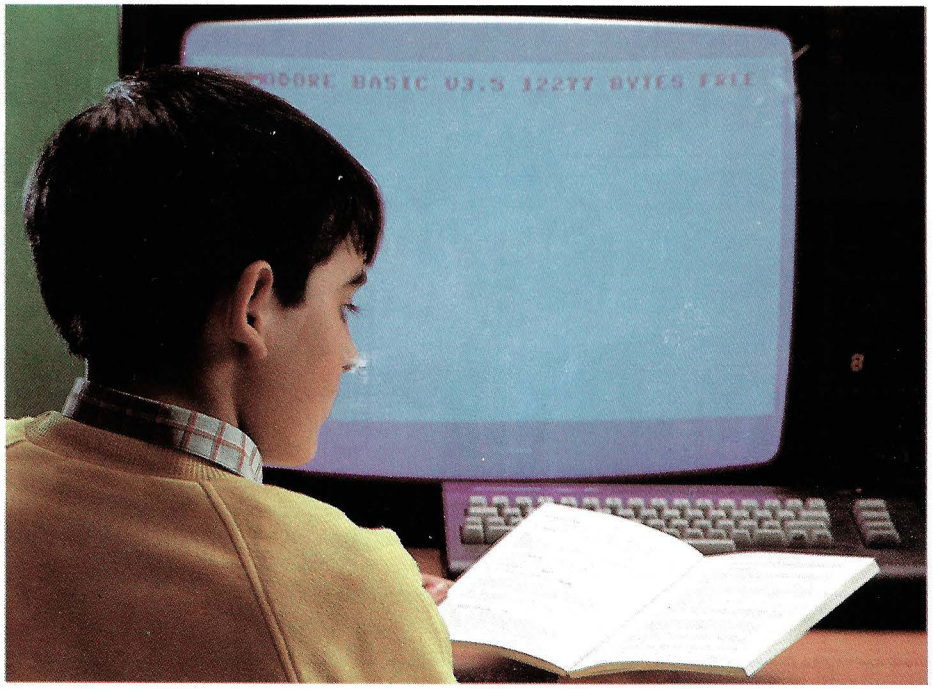
3. Test de control.
Una vez finalizada la exposición de un capítulo, el programa interrogará al usuario para que él mismo llegue a decidir si sus conocimientos son suficientes como para pasar a otro capítulo. A la hora de confeccionar las preguntas de control, el programa debe utilizar procedimientos pseudoaleatorios para no repetir siempre las mismas cuestiones; también garantizar que las preguntas sean variadas y significativas, de forma que de su contestación correcta o incorrecta se pueda deducir un resultado final.

- *Programas de apoyo*

La "filosofía" de esta categoría del software didáctico es completamente distinta. En este caso el programa servirá de complemento a las exposiciones realizadas por un profesor humano. Por lo tanto, no resultará imprescindible que su documentación y sencillez de manejo sean tan escrupulosas como en el caso de los programas de autoestudio.
El objetivo fundamental de estos programas radica en complementar la labor del profesor. Generalmente, suelen hacer especial hincapié en ejercicios prácticos que sirvan para estudiar la casuística que puede presentarse sobre las lecciones teóricas. Un factor importante, al igual que en el caso anterior, es la aleatoriedad en las preguntas realizadas al alumno.
Una condición imprescindible para el eficaz aprovechamiento de este tipo de programas reside en el perfecto conocimiento que deben tener los profesores sobre su funcionamiento. Resulta de vital importancia que, por un lado el profesor vea en el programa a un ayudante y no a un competidor, y por otro que el alumno no magnifique al programa y, en consecuencia, deje de valorar al profesor. Para conseguir estos dos objetivos es fundamental partir de una correcta exposición sobre qué es un programa de ordenador, cuáles son sus ventajas y dónde están sus limitaciones.

- *Programas de educación informática*

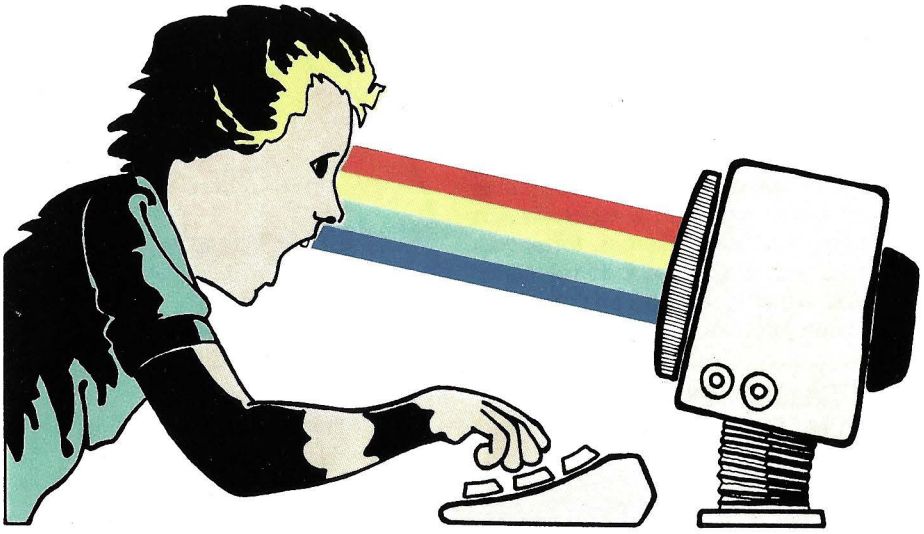
Los dos grupos anteriores de programas pueden prestar sus servicios sobre prácticamente cualquier materia: Historia, Geografía, Matemáticas, Química, Idiomas, etc. Una de estas materias es precisamente la Informática, y dado que en este caso las materias a estudiar son las que soportan al programa educativo, vamos a englobarlas en un grupo particular. Por supuesto, dentro de los programas de educación informática encontraremos algunos de tipo autoestudio y otros de tipo apoyo. Si en cualquier caso la capacidad docente de un ordenador puede ser explotada mediante programas educativos, parece obvio que una de las materias que



Forman parte del software educativo todos aquellos programas cuya misión fundamental es instruir o entrenar al usuario sobre alguna materia susceptible de estudio.

más se presta a un estudio mecanizado es precisamente la Informática. Una de las características principales de este tipo de programas consiste en que el ordenador se utiliza a sí mismo, tanto para realizar exposiciones teóricas, como para los ejercicios prácticos.

VENTAJAS ADICIONALES



En los programas educativos resulta de vital importancia el uso de efectos especiales que atraigan la atención del alumno.

Resulta evidente que la utilización del software educativo en cualquiera de su tres posibilidades, ofrece una nueva vía a los profesionales de la educación. En este sentido, cabe afirmar que el conjunto formado por el ordenador más el programa es una herramienta pedagógica que, poco a poco, se va imponiendo en los planes de estudio. Del uso de la Informática aplicada a la enseñanza se obtienen dos ventajas adicionales:

1. El alumno situado frente a una máquina tiene que utilizar todo su ingenio para obtener no sólo buenas calificaciones, sino también un dominio sobre ella. Evidentemente, la única forma de lograr este objetivo es desarrollando la capacidad de razonamiento lógico, que más adelante podrá ser utilizada en cualquier tipo de actividad. 2. Como consecuencia de la utilización del sistema informático para el estudio, los usuarios terminan captando

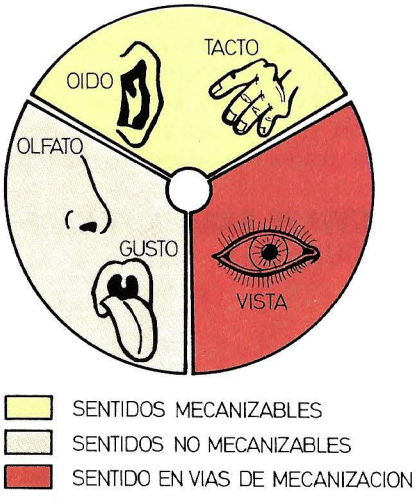
Periféricos polivalentes

Una de las facetas que más ha evolucionado en el mundo del ordenador es la comunicación entre el hombre y la máquina. Las mejoras conseguidas en este campo han afectado tanto a los periféricos de entrada de datos como a los de salida. Por ejemplo, del empleo masivo de la tarjeta perforada como soporte de entrada, se ha pasado a utilizar medios más cómodos e interactivos como puede ser el teclado. En el mismo orden, del trabajo con voluminosas y lentas impresoras, se ha pasado a pequeñas y rápidas impresoras de sobremesa.
Dentro de los periféricos de comunicación existe la tendencia a unificar en un único equipo múltiples capacidades de diálogo. Una de las unidades más polivalentes en la actualidad es el terminal que, en algunos casos, llega a sintetizar hasta cinco distintas unidades funcionales:

- TECLADO
Sirve como elemento de comunicación de entrada; esto es: el usuario "pasa" información al ordenador a través del mismo.

- PANTALLA
Complementa al teclado encargándose de las comunicaciones de salida; para ello visualiza los mensajes y datos en juego.

- SEÑALES AUDIBLES
También como elemento de salida de información, los terminales disponen de un juego de sonidos. Estos suelen utilizarse para hacer especial énfasis en determinados tipos de comunicaciones.



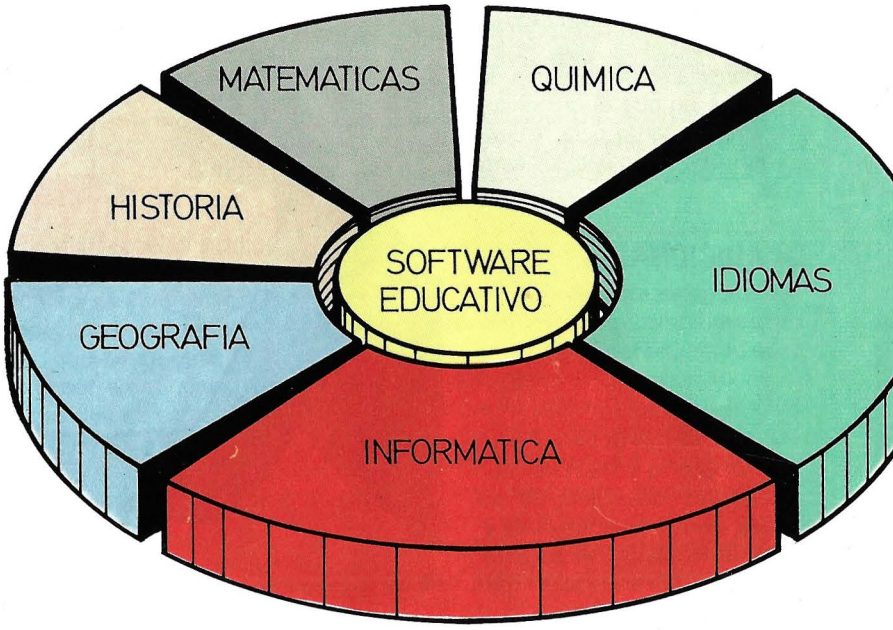
- PANTALLAS SENSIBLES
En algunos casos, además de utilizar la pantalla para la salida de información, ésta también puede emplearse en operaciones de entrada. Estas pantallas deben ser capaces de "sentir" el contacto humano (generalmente, apoyando un dedo), en orden a comunicar al programa cuál de las opciones presentes en la pantalla debe ser activada.

- COMUNICACION EN LENGUAJE NATURAL
Por último, podemos citar la existencia de terminales capaces de comunicarse de viva voz: entienden determinadas órdenes dictadas verbalmente por el usuario, y contestan también en forma hablada (sintetizando la voz).
El uso de terminales sofisticadas puede resultar útil en muchos casos. Sin duda, uno de los más claros, reside en la utilización del ordenador con fines didácticos. Combinando un buen programa educativo con un buen periférico los resultados pueden ser espectaculares.

el significado real del concepto ordenador; ello, sin duda alguna, le resultará de gran importancia cuando en el futuro se encuentre con la necesidad de utilizar programas y ordenadores.
Sin olvidar su objetivo fundamental de profundizar en el estudio de una materia concreta, cabe afirmar que los paquetes educativos contribuyen a racionalizar la adquisición de conocimientos, forzando a que el estudiante razone y no memorice y, adicionalmente, aportan una cierta formación en Informática.

PROGRAMAS EDUCATIVOS ESPECIALES

Hasta ahora nos hemos limitado a describir las características del software educativo aplicado a la enseñanza general.
Fundamentalmente, este tipo de software está dedicado a alumnos adolescentes, aunque, por supuesto, también puede ser utilizado por personas de cualquier edad. No obstante, existen ciertos programas educativos especiales que se utilizan para la formación de profesionales altamente cualificados. Tal vez el ejemplo más típico de programas educativos especiales se encuentre en los dedicados a la formación de pilotos comerciales o militares. Además de recortar los gastos económicos que derivarían de utilizar aviones auténticos, se evitan los riesgos altamente peligrosos que correrían los materiales educativos al ponerse en manos de un alumno inexperto y, por supuesto, se garantiza la integridad física del propio alumno.
Es importante no confundir los programas educativos especiales con los juegos de simulación. Mientras los primeros persiguen (siguiendo el mismo ejemplo) formar auténticos pilotos, los segundos se limitan a entretener y divertir. En consecuencia, los simuladores didácticos deben reflejar con exquisita precisión las características del modelo real, sin preocuparse demasiado de los efectos especiales. En cambio, los juegos de simulación se vuelcan precisamente en los efectos especiales, y tan sólo contemplan una reproducción aproximada del modelo real.



Las materias susceptibles de ser estudiadas mediante software educativo son prácticamente inagotables. La figura muestra las disciplinas actualmente más afectadas por esta técnica de estudio.

Fases convencionales en las que se desarrolla un programa educativo.

Aunque el ejemplo anterior resulta suficientemente ilustrativo, pueden citarse otros muchos temas que en la actualidad se apoyan en software educativo para la formación de profesionales; en general, todos aquellos que suponen el uso de medios contaminantes o peligrosos. Entre estas actividades podemos citar un grupo, hasta ahora olvidado, pero que en la actualidad se está desarrollando rápidamente: programas educativos médicos.

HARDWARE PARA SOFTWARE DE EDUCACION

Hace algunos años los distribuidores de productos informáticos "apostaron" por una solución compacta que consistía e uso de un ordenador especializado ún mente en procesos didácticos. En la tualidad esta opción está prácticame olvidada; los modernos microordenado tienen suficiente capacidad como para mitir programas educativos de carác general, esta última solución permite a más utilizar el equipo para otras fi dades.
Si el propietario del ordenador es un c gio, además de utilizarlo para formar a alumnos, puede disponer del mismo p automatizar la gestión del colegio.
Desde luego, no es generalizable la mación de que para ejecutar un progra educativo de carácter general sea nece rio utilizar ordenadores especiales. embargo, cuando el objetivo del progra es la formación o entrenamiento de pr sionales, puede ser necesaria la partic ción de un ordenador especializado e materia objeto del entrenamiento.

2

SOFTWARE

Ingelek